Computer science

Python and Latex

2020 – 2021

Lama TARSISSI

Outline

Session I

1. Introduction
2. Installation and configuration
3. Simple examples

1.Introduction

What is Python?

In 1989 the dutch computer scientist, Guido van Rossum, created PYTHON .

What is Python?

In 1989 the dutch computer scientist, Guido van Rossum, created PYTHON .

- The name of this language comes from the TV-series "Monty Python's Flying Circus" of which G. van Rossum is a big fan.

What is Python?

In 1989 the dutch computer scientist, Guido van Rossum, created PYTHON .

- The name of this language comes from the TV-series "Monty Python's Flying Circus" of which G. van Rossum is a big fan.
- The first version was published in 1991.

What is Python?

In 1989 the dutch computer scientist, Guido van Rossum, created PYTHON .

- The name of this language comes from the TV-series "Monty Python's Flying Circus" of which G. van Rossum is a big fan.
- The first version was published in 1991.
- The most recent version is 3.8 that was released in 14/10/2019.

What is Python?

In 1989 the dutch computer scientist, Guido van Rossum, created PYTHON .

- The name of this language comes from the TV-series "Monty Python's Flying Circus" of which G. van Rossum is a big fan.
- The first version was published in 1991.
- The most recent version is 3.8 that was released in 14/10/2019.
- *Python Software Foundation* is the association that organizes the developement of this language and manages the community of developers and users.

This programming language presents several interesting caracteristics:

1. It is multiplateform. This means that it can be used on several operating systems: Windows, Mac OS X, Linux, Android, iOS, starting from the Mini-computers Raspberry Pi to the latest super calculators.

This programming language presents several interesting caracteristics:

1. It is multiplateform. This means that it can be used on several operating systems: Windows, Mac OS X, Linux, Android, iOS, starting from the Mini-computers Raspberry Pi to the latest super calculators.
2. It is free. It can be installed over any PC that you want and even on your phones.

This programming language presents several interesting caracteristics:

1. It is multiplateform. This means that it can be used on several operating systems: Windows, Mac OS X, Linux, Android, iOS, starting from the Mini-computers Raspberry Pi to the latest super calculators.
2. It is free. It can be installed over any PC that you want and even on your phones.
3. It is a high level language. It does NOT require the user to have a big knowledge in the functionning of the PC.

This programming language presents several interesting caracteristics:

1. It is multiplateform. This means that it can be used on several operating systems: Windows, Mac OS X, Linux, Android, iOS, starting from the Mini-computers Raspberry Pi to the latest super calculators.
2. It is free. It can be installed over any PC that you want and even on your phones.
3. It is a high level language. It does NOT require the user to have a big knowledge in the functionning of the PC.
4. It is an interpretated language. This means that a Python script doesn't need to be compiled to be executed. Contrarily to other languages like C and C++.

This programming language presents several interesting caracteristics:

1. It is multiplateform. This means that it can be used on several operating systems: Windows, Mac OS X, Linux, Android, iOS, starting from the Mini-computers Raspberry Pi to the latest super calculators.
2. It is free. It can be installed over any PC that you want and even on your phones.
3. It is a high level language. It does NOT require the user to have a big knowledge in the functionning of the PC.
4. It is an interpretated language. This means that a Python script doesn't need to be compiled to be executed. Contrarily to other languages like C and C++.
5. It is object oriented. This means that it is possible to see in Python some entities that mime the real world (a cell, a protein, an atom, etc.) with a certain number of functionning rules and interactions.

This programming language presents several interesting caracteristics:

1. It is multiplateform. This means that it can be used on several operating systems: Windows, Mac OS X, Linux, Android, iOS, starting from the Mini-computers Raspberry Pi to the latest super calculators.
2. It is free. It can be installed over any PC that you want and even on your phones.
3. It is a high level language. It does NOT require the user to have a big knowledge in the functionning of the PC.
4. It is an interpretated language. This means that a Python script doesn't need to be compiled to be executed. Contrarily to other languages like C and C++.
5. It is object oriented. This means that it is possible to see in Python some entities that mime the real world (a cell, a protein, an atom, etc.) with a certain number of functionning rules and interactions.
6. It is relatively simple to work with it.

This programming language presents several interesting caracteristics:

1. It is multiplateform. This means that it can be used on several operating systems: Windows, Mac OS X, Linux, Android, iOS, starting from the Mini-computers Raspberry Pi to the latest super calculators.
2. It is free. It can be installed over any PC that you want and even on your phones.
3. It is a high level language. It does NOT require the user to have a big knowledge in the functionning of the PC.
4. It is an interpretated language. This means that a Python script doesn't need to be compiled to be executed. Contrarily to other languages like C and C++.
5. It is object oriented. This means that it is possible to see in Python some entities that mime the real world (a cell, a protein, an atom, etc.) with a certain number of functionning rules and interactions.
6. It is relatively simple to work with it.
7. It is used in several domains, like bioinformatics, data analysis etc..

This programming language presents several interesting caracteristics:

1. It is multiplateform. This means that it can be used on several operating systems: Windows, Mac OS X, Linux, Android, iOS, starting from the Mini-computers Raspberry Pi to the latest super calculators.
2. It is free. It can be installed over any PC that you want and even on your phones.
3. It is a high level language. It does NOT require the user to have a big knowledge in the functionning of the PC.
4. It is an interpretated language. This means that a Python script doesn't need to be compiled to be executed. Contrarily to other languages like C and C++.
5. It is object oriented. This means that it is possible to see in Python some entities that mime the real world (a cell, a protein, an atom, etc.) with a certain number of functionning rules and interactions.
6. It is relatively simple to work with it.
7. It is used in several domains, like bioinformatics, data analysis etc..

All theses caracteristics make of PYTHON a very useful language.
That's why, nowadays it is used in high schools and higher education levels.

2.Installation and configuration

To be a good programmer, you need to do lot of tests and experiments.

To be a good programmer, you need to do lot of tests and experiments.

==> You need to install PYTHON on your PC

To be a good programmer, you need to do lot of tests and experiments.

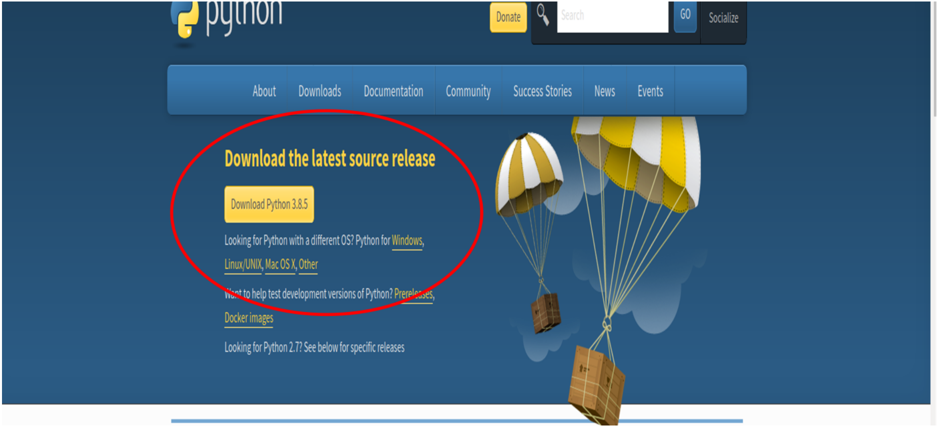
==> You need to install PYTHON on your PC

With Windows, Mac or Linux OS, it is easily and freely installed on your PC.

To be a good programmer, you need to do lot of tests and experiments.

==> You need to install PYTHON on your PC

With Windows, Mac or Linux OS, it is easily and freely installed on your PC.



With Windows

```
PS C:\Users\pierre> python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

With MAC-OS

```
iMac-de-pierre:Downloads$ python
Python 3.7.1 (default, Dec 14 2018, 19:28:38)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

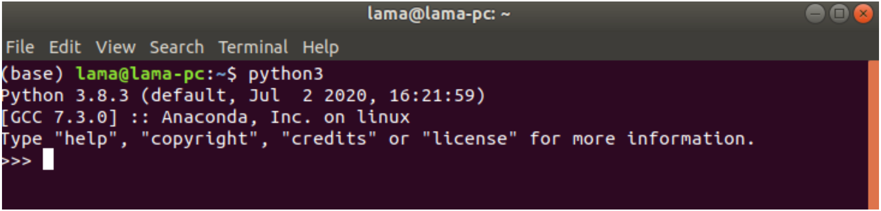
With Windows

```
PS C:\Users\pierre> python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

With MAC-OS

```
iMac-de-pierre:Downloads$ python
Python 3.7.1 (default, Dec 14 2018, 19:28:38)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

With LINUX



Notations

The standard **prompt** for the interactive mode is >>> called triple right chevron. Once you see these characters, you'll know you are in PYTHON.

Notations

The standard **prompt** for the interactive mode is >>> called triple right chevron.
Once you see these characters, you'll know you are in PYTHON.
Now, you can write and run Python code as you wish.

Notations

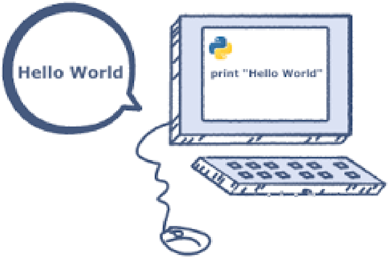
The standard **prompt** for the interactive mode is >>> called triple right chevron.
Once you see these characters, you'll know you are in PYTHON.
Now, you can write and run Python code as you wish.
Note that when you close the session, your code will be gone.

Notations

The standard **prompt** for the interactive mode is >>> called triple right chevron.
Once you see these characters, you'll know you are in PYTHON.
Now, you can write and run Python code as you wish.
Note that when you close the session, your code will be gone.**BUT!!**
When you work interactively, every expression and statement you type in is evaluated and executed immediately.

Notations

The standard **prompt** for the interactive mode is >>> called triple right chevron.
Once you see these characters, you'll know you are in PYTHON.
Now, you can write and run Python code as you wish.
Note that when you close the session, your code will be gone.**BUT!!**
When you work interactively, every expression and statement you type in is evaluated and executed immediately.



Notations

The standard **prompt** for the interactive mode is >>> called triple right chevron.
Once you see these characters, you'll know you are in PYTHON.
Now, you can write and run Python code as you wish.
Note that when you close the session, your code will be gone.**BUT!!**
When you work interactively, every expression and statement you type in is evaluated and executed immediately.

Python

```
>>> print('Hello World!')
Hello World!
>>> 2 + 5
7
>>> print('Welcome to Real Python!')
Welcome to Real Python!
```

An easier way to deal with PYTHON is to pass through ANACONDA .

Anaconda-navigator

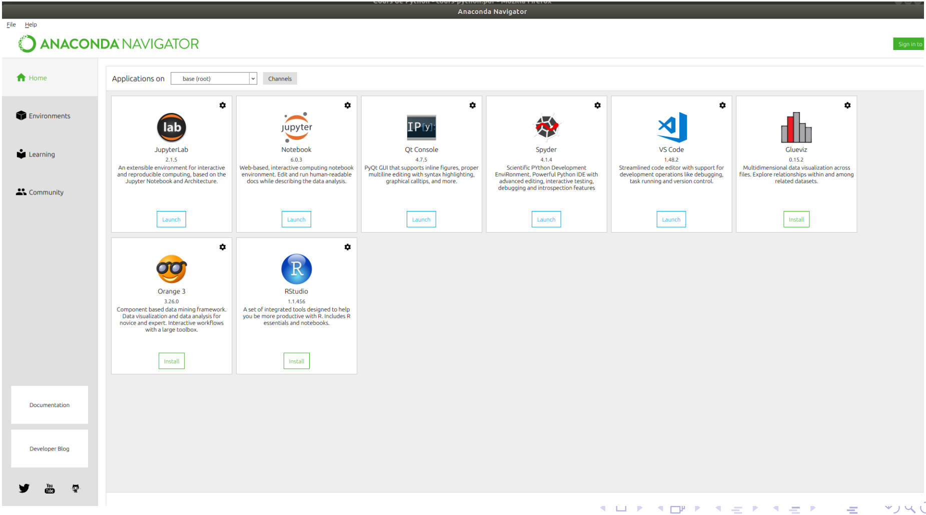
An easier way to deal with PYTHON is to pass through ANACONDA .
It is a free distribution and an open programming source for Python and R.

An easier way to deal with PYTHON is to pass through ANACONDA .
It is a free distribution and an open programming source for Python and R.
To do so, you should follow this link to install it:

https://docs.anaconda.com/anaconda/install/

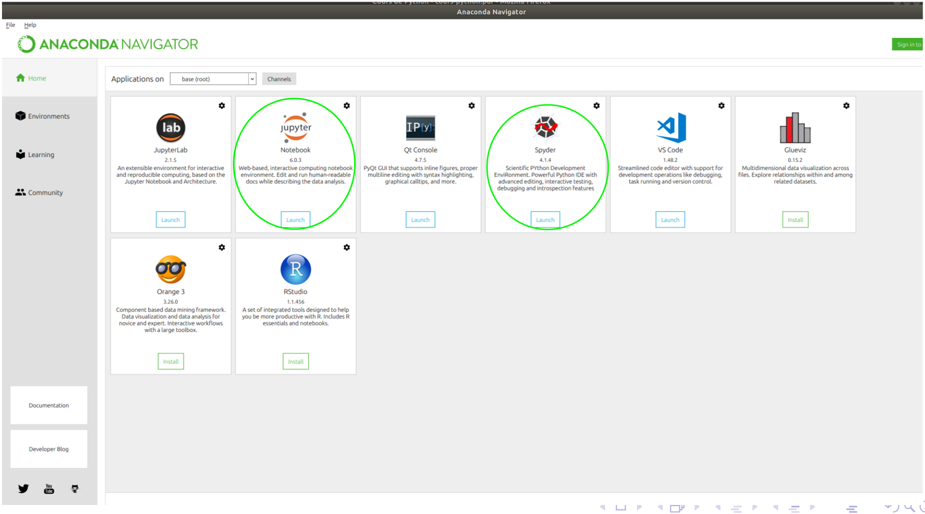
An easier way to deal with PYTHON is to pass through ANACONDA .
It is a free distribution and an open programming source for Python and R.
To do so, you should follow this link to install it:

https://docs.anaconda.com/anaconda/install/

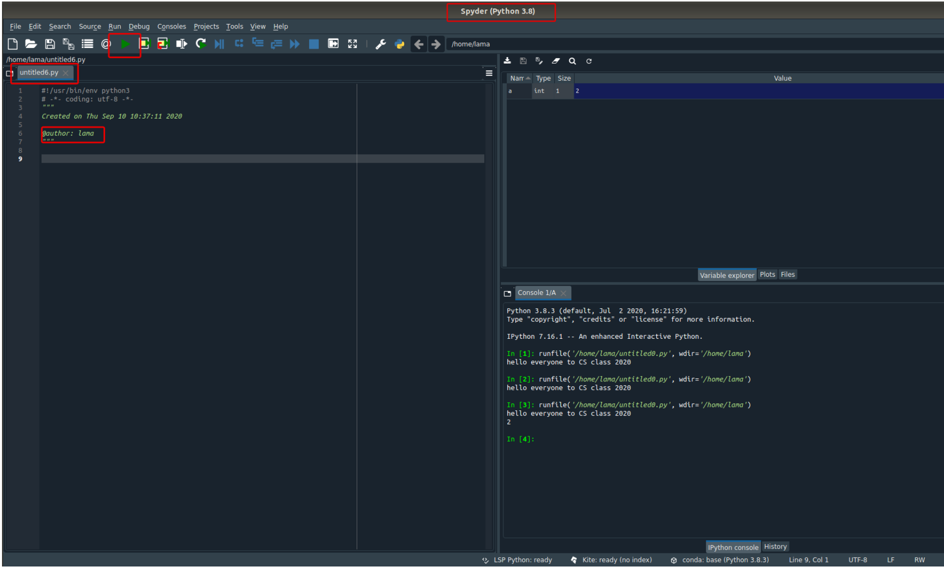


An easier way to deal with PYTHON is to pass through ANACONDA .
It is a free distribution and an open programming source for Python and R.
To do so, you should follow this link to install it:

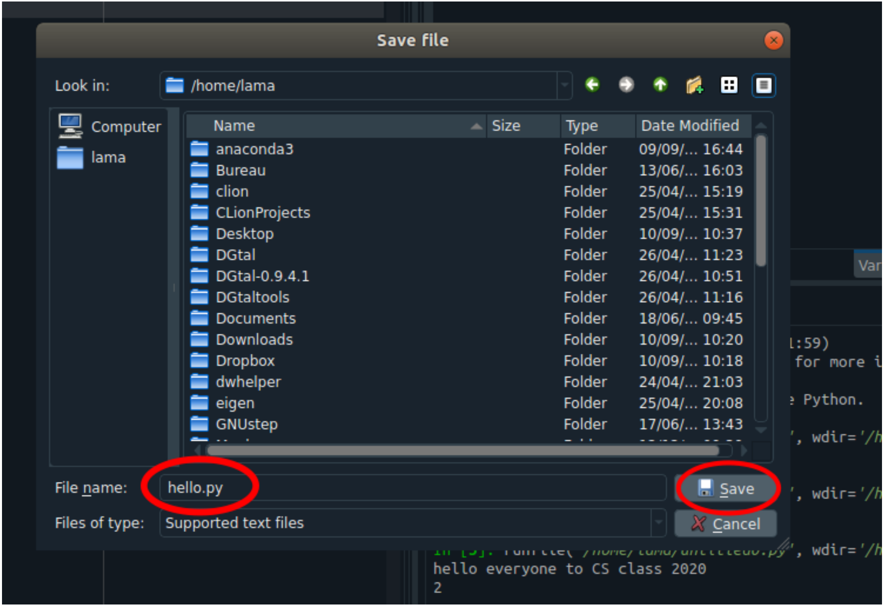
https://docs.anaconda.com/anaconda/install/



Spyder

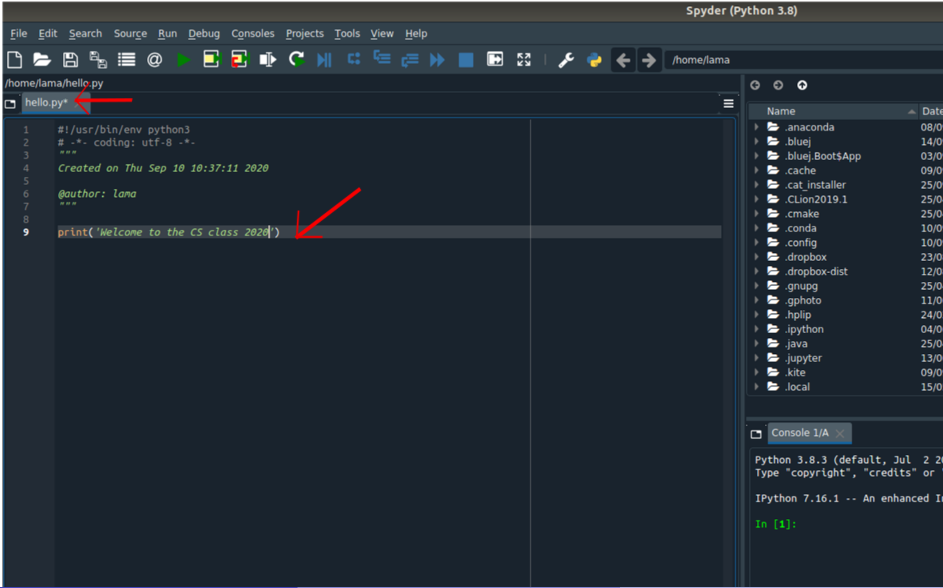

Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/lama
/home/lama/untitled6.py
untitled6.py
#!/usr/bin/env python3
-*- coding: utf-8 -*-
"""
Created on Thu Sep 10 10:37:11 2020
@author: lama
"""
Name Type Size Value
a int 1 2
Variable explorer Plots Files
Console 1/A
Python 3.8.3 (default, Jul 2 2020, 16:21:59)
Type "copyright", "credits" or "license" for more information.
IPython 7.16.1 -- An enhanced Interactive Python.
In [1]: runfile('/home/lama/untitled0.py', wdir='/home/lama')
hello everyone to CS class 2020
In [2]: runfile('/home/lama/untitled0.py', wdir='/home/lama')
hello everyone to CS class 2020
In [3]: runfile('/home/lama/untitled0.py', wdir='/home/lama')
hello everyone to CS class 2020
2
In [4]:
IPython console History
LSP Python: ready
Kite: ready (no index)
conda: base (Python 3.8.3)
Line 9, Col 1
UTF-8
LF
RW

Spyder

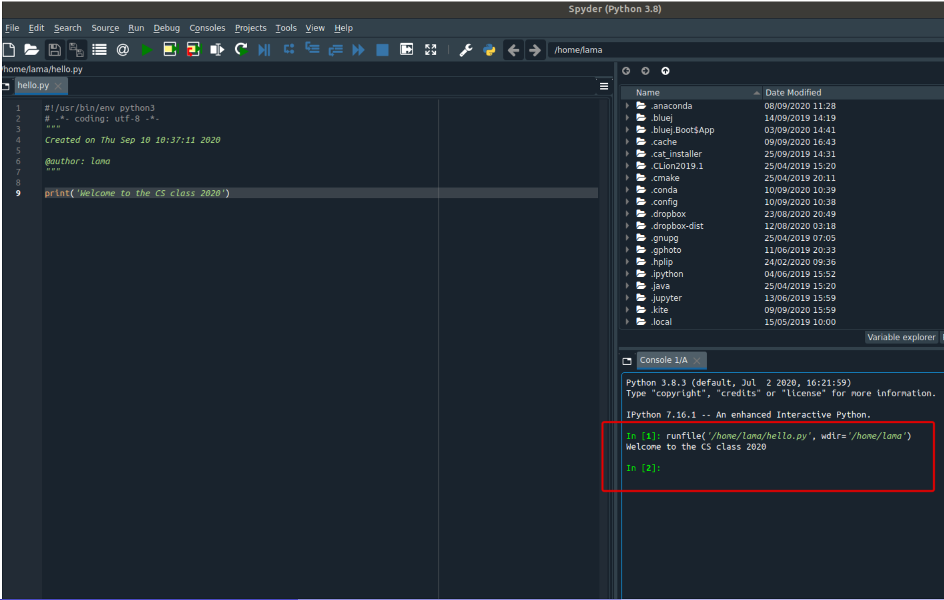


Save file
Look in:
/home/lama
Computer
lama
Name
Size
Type
Date Modified
anaconda3
Folder
09/09/... 16:44
Bureau
Folder
13/06/... 16:03
clion
Folder
25/04/... 15:19
CLionProjects
Folder
25/04/... 15:31
Desktop
Folder
10/09/... 10:37
DGtal
Folder
26/04/... 11:23
DGtal-0.9.4.1
Folder
26/04/... 10:51
DGtaltools
Folder
26/04/... 11:16
Documents
Folder
18/06/... 09:45
Downloads
Folder
10/09/... 10:20
Dropbox
Folder
10/09/... 10:18
dwhelper
Folder
24/04/... 21:03
eigen
Folder
25/04/... 20:08
GNUstep
Folder
17/06/... 13:43
File name:
hello.py
Save
Files of type:
Supported text files
Cancel
hello everyone to CS class 2020
2

Spyder


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/lama
/home/lama/hello.py
hello.py*
#!/usr/bin/env python3
-*- coding: utf-8 -*-
"""
Created on Thu Sep 10 10:37:11 2020
@author: lama
"""
print('Welcome to the CS class 2020')
Name
.anaconda
.bluej
.bluej.Boot$App
.cache
.cat_installer
.CLion2019.1
.cmake
.conda
.config
.dropbox
.dropbox-dist
.gnupg
.gphoto
.hplip
.ipython
.java
.jupyter
.kite
.local
Console 1/A
IPython 7.16.1 -- An enhanced
In [1]:

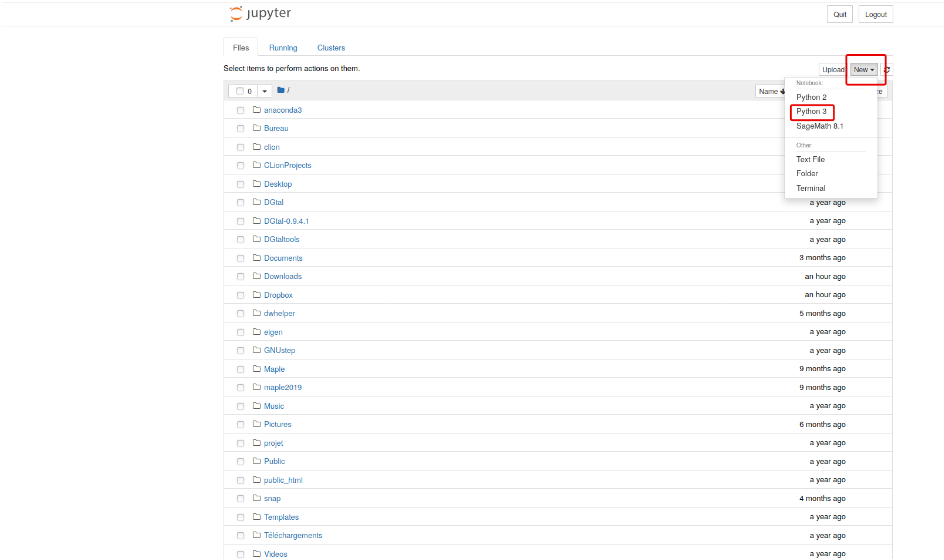
Spyder


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/lama
/home/lama/hello.py
hello.py
#!/usr/bin/env python3
-*- coding: utf-8 -*-
"""
Created on Thu Sep 10 10:37:11 2020

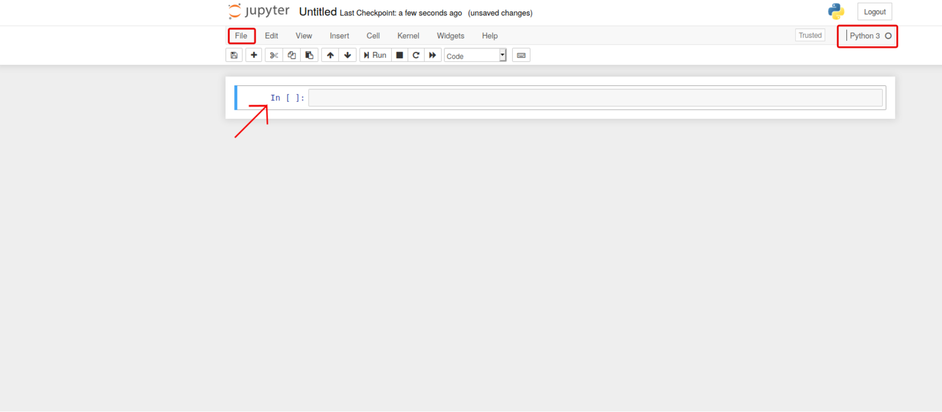
@author: lama
"""

print('Welcome to the CS class 2020')
Name
Date Modified
.anaconda 08/09/2020 11:28
.bluej 14/09/2019 14:19
.bluej.Boot$App 03/09/2020 14:41
.cache 09/09/2020 16:43
.cat_installer 25/09/2019 14:31
.CLion2019.1 25/04/2019 15:20
.cmake 25/04/2019 20:11
.conda 10/09/2020 10:39
.config 10/09/2020 10:38
.dropbox 23/08/2020 20:49
.dropbox-dist 12/08/2020 03:18
.gnupg 25/04/2019 07:05
.gphoto 11/06/2019 20:33
.hplip 24/02/2020 09:36
.ipython 04/06/2019 15:52
.java 25/04/2019 15:20
.jupyter 13/06/2019 15:59
.kite 09/09/2020 15:59
.local 15/05/2019 10:00
Variable explorer
Console 1/A
Python 3.8.3 (default, Jul 2 2020, 16:21:59)
Type "copyright", "credits" or "license" for more information.
IPython 7.16.1 -- An enhanced Interactive Python.
In [1]: runfile('/home/lama/hello.py', wdir='/home/lama')
Welcome to the CS class 2020
In [2]:

Jupyter

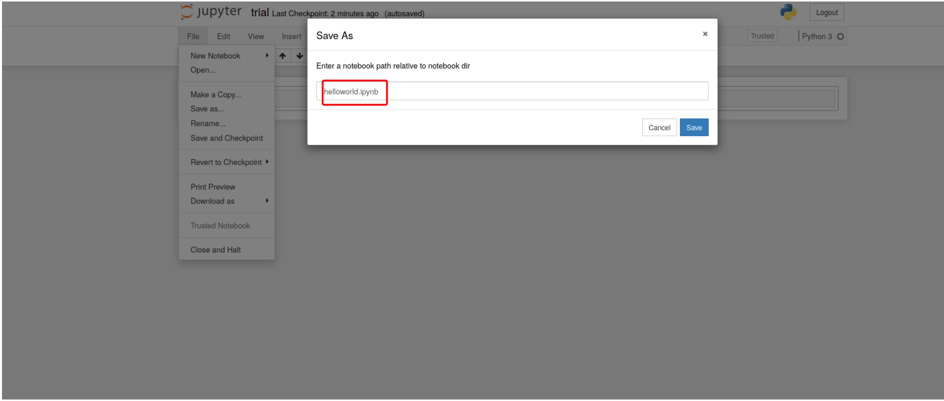

jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/
Name
Notebook:
Python 2
Python 3
SageMath 8.1
Other:
Text File
Folder
Terminal
anaconda3
Bureau
clion
CLionProjects
Desktop
DGtal
a year ago
DGtal-0.9.4.1
a year ago
DGtaltools
a year ago
Documents
3 months ago
Downloads
an hour ago
Dropbox
an hour ago
dwhelper
5 months ago
eigen
a year ago
GNUstep
a year ago
Maple
9 months ago
maple2019
9 months ago
Music
a year ago
Pictures
6 months ago
projet
a year ago
Public
a year ago
public_html
a year ago
snap
4 months ago
Templates
a year ago
Téléchargements
a year ago
Videos
a year ago

Jupyter



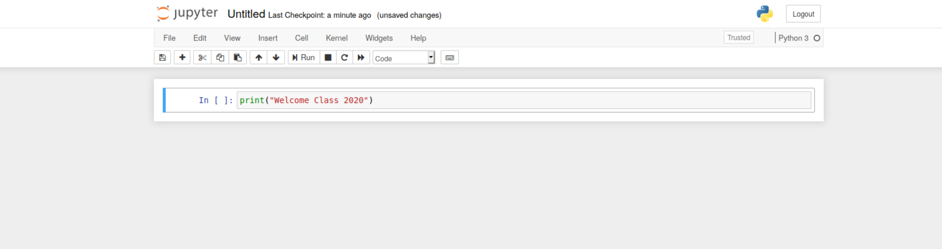
jupyter
Untitled Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
In [ ]:

Jupyter

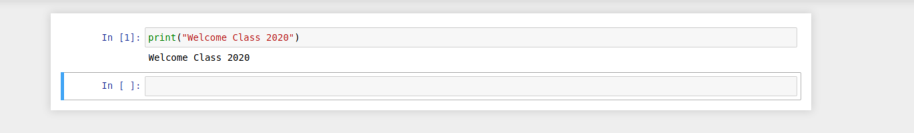


jupyter trial Last Checkpoint: 2 minutes ago (autosaved)
Logout
File Edit View Insert
Trusted
Python 3
New Notebook
Open...
Make a Copy...
Save as...
Rename...
Save and Checkpoint
Revert to Checkpoint
Print Preview
Download as
Trusted Notebook
Close and Halt
Save As
Enter a notebook path relative to notebook dir
helloworld.ipynb
Cancel
Save

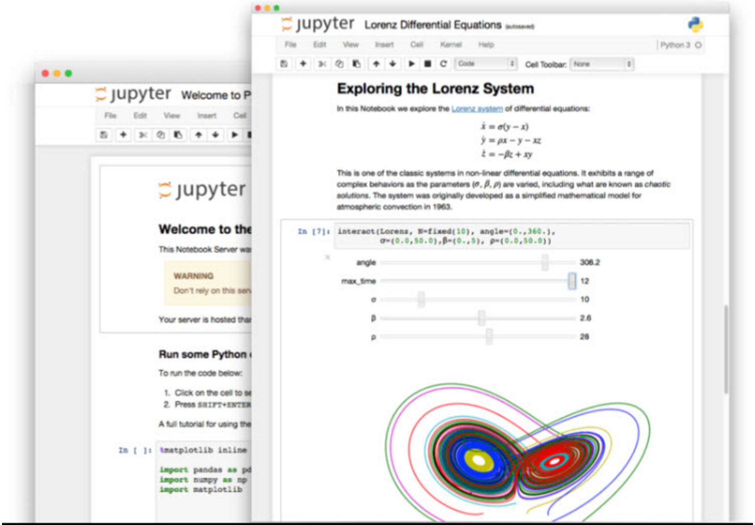
Jupyter


jupyter
Untitled Last Checkpoint: a minute ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
In [ ]: print("Welcome Class 2020")

Jupyter



Embeded Animation



Jupyter Notebooks via https://jupyter.org/

Outline

Session II

1. Bugs
2. Types
3. Built-in functions
4. Operations
5. Simple examples

1.Bugs

History

The term **"bug"** was used in an account by computer pioneer Grace Hopper, who publicized the cause of a malfunction in an early electromechanical computer. A typical version of the story is:

Grace Murray Hopper

History

The term **"bug"** was used in an account by computer pioneer Grace Hopper, who publicized the cause of a malfunction in an early electromechanical computer. A typical version of the story is:

Remark

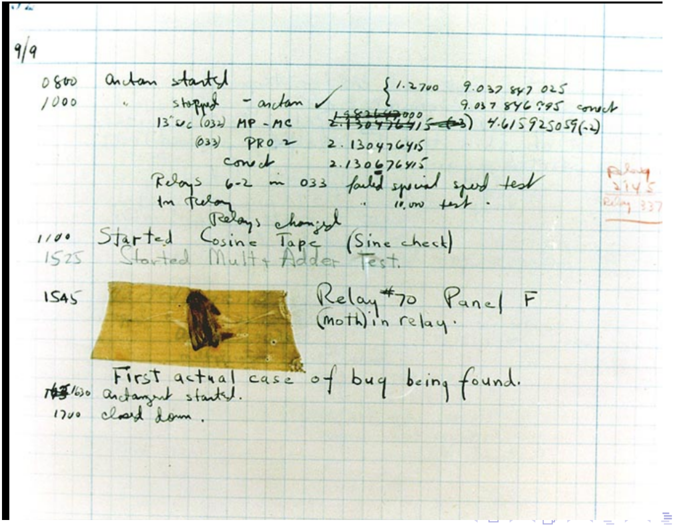
In 1946, when Hopper was released from active duty, she joined the Harvard Faculty at the Computation Laboratory where she continued her work on the Mark II and Mark III. Operators traced an error in the Mark II to a moth trapped in a relay, coining the term bug. This bug was carefully removed and taped to the log book. Stemming from the first bug, today we call errors or glitches in a program a bug.

Moths

Emperor gum moth, *Opodiphthera eucalypti*

History

The term **"bug"** was used in an account by computer pioneer Grace Hopper, who publicized the cause of a malfunction in an early electromechanical computer. A typical version of the story is:



Bugs

As defined in Wikipedia:

Definition

"An **error** is a deviation from accuracy or correctness" and "A **software bug** is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways".

Bugs

As defined in Wikipedia:

Definition

"An **error** is a deviation from accuracy or correctness" and "A **software bug** is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways".

Software bugs are of many types.

A **bug** is a bug no matter what.

Bugs

As defined in Wikipedia:

Definition

"An **error** is a deviation from accuracy or correctness" and "A **software bug** is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways".

Software bugs are of many types. A **bug** is a bug no matter what.
But sometimes, it is important to understand the nature, its implications and the cause to process it better.

Bugs

As defined in Wikipedia:

Definition

"An **error** is a deviation from accuracy or correctness" and "A **software bug** is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways".

Software bugs are of many types. A **bug** is a bug no matter what.
But sometimes, it is important to understand the nature, its implications and the cause to process it better.

This helps for faster reaction and most importantly, appropriate reaction.

There are 7 Common Types of Software Errors (Bugs) That Every Tester Should Know

There are 7 Common Types of Software Errors (Bugs) That Every Tester Should Know

1. Functionality Errors: If something that you expect it to do is hard, awkward, confusing, or impossible.

There are 7 Common Types of Software Errors (Bugs) That Every Tester Should Know

1. Functionality Errors: If something that you expect it to do is hard, awkward, confusing, or impossible.
2. Communication Errors: Anything that the end user needs to know in order to use the software should be made available on screen.

There are 7 Common Types of Software Errors (Bugs) That Every Tester Should Know

1. Functionality Errors: If something that you expect it to do is hard, awkward, confusing, or impossible.
2. Communication Errors: Anything that the end user needs to know in order to use the software should be made available on screen.
3. Missing command Errors: When an expected command is missing.

There are 7 Common Types of Software Errors (Bugs) That Every Tester Should Know

1. Functionality Errors: If something that you expect it to do is hard, awkward, confusing, or impossible.
2. Communication Errors: Anything that the end user needs to know in order to use the software should be made available on screen.
3. Missing command Errors: When an expected command is missing.
4. Syntactic Error: They are misspelled words or grammatically incorrect sentences.

5. Error handling Errors: Any errors that occur while the user is interacting with the software needs to be handled in a clear and meaningful manner. If NOT, it is called as an Error Handling Error.

5. Error handling Errors: Any errors that occur while the user is interacting with the software needs to be handled in a clear and meaningful manner. If NOT, it is called as an Error Handling Error.
5. Calculation Errors: These errors occur due to any of the following reasons: Bad logic, Incorrect formulae, Data type mismatch, Coding errors, Function call issues, etc.

5. Error handling Errors: Any errors that occur while the user is interacting with the software needs to be handled in a clear and meaningful manner. If NOT, it is called as an Error Handling Error.
5. Calculation Errors: These errors occur due to any of the following reasons: Bad logic, Incorrect formulae, Data type mismatch, Coding errors, Function call issues, etc.
5. Control flow errors: It describes what it will do next and on what condition.

5. Error handling Errors: Any errors that occur while the user is interacting with the software needs to be handled in a clear and meaningful manner. If NOT, it is called as an Error Handling Error.
5. Calculation Errors: These errors occur due to any of the following reasons: Bad logic, Incorrect formulae, Data type mismatch, Coding errors, Function call issues, etc.
5. Control flow errors: It describes what it will do next and on what condition.

In Python, the most famous Bugs are:

- Syntax: bugs in structure of the input;
- Routine: bugs in execution;
- Semantic: the program is correct in syntax and execution, but the output is not we expected.

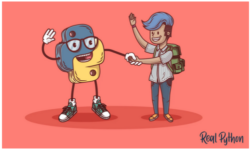
Examples

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

2.Types

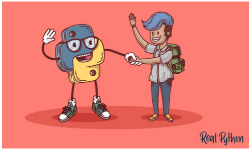
Basic Data Types in Python

It's time to dig into the Python language.
First up is a discussion of the basic data types that are built into Python.



Basic Data Types in Python

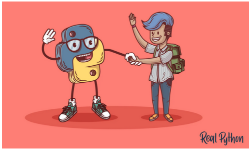
It's time to dig into the Python language.
First up is a discussion of the basic data types that are built into Python.



There are several basic **numeric, string, and Boolean** types that are built into Python.

Basic Data Types in Python

It's time to dig into the Python language.
First up is a discussion of the basic data types that are built into Python.



There are several basic **numeric, string, and Boolean** types that are built into Python.

We will see some **Python's built-in functions** like the built-in print()function.

Integers

In Python 3, there is effectively no limit to how **long** an integer value can be. Of course, it is constrained by the amount of memory your system has.

Integers

In Python 3, there is effectively no limit to how **long** an integer value can be. Of course, it is constrained by the amount of memory your system has.

Python >>>

```
>>> print(123123123123123123123123123123123123123123123 + 1)
123123123123123123123123123123123123123123124
```

Integers

In Python 3, there is effectively no limit to how **long** an integer value can be. Of course, it is constrained by the amount of memory your system has.

Python >>>

```
>>> print(123123123123123123123123123123123123123123123123 + 1)
123123123123123123123123123123123123123123123124
```

Python interprets a sequence of decimal digits without any prefix to be a decimal number:

Integers

In Python 3, there is effectively no limit to how **long** an integer value can be. Of course, it is constrained by the amount of memory your system has.

Python >>>

```
>>> print(123123123123123123123123123123123123123123123123 + 1)
123123123123123123123123123123123123123123123124
```

Python interprets a sequence of decimal digits without any prefix to be a decimal number:

Python >>>

```
>>> print(10)
10
```

Floating-Point Numbers

1. The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character **e** or **E** followed by a positive or negative integer may be appended to specify scientific notation:

Floating-Point Numbers

1. The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character **e** or **E** followed by a positive or negative integer may be appended to specify scientific notation:

Python

```
>>> 4.2
4.2
>>> type(4.2)
<class 'float'>
>>> 4.
4.0
>>> .2
0.2

>>> .4e7
4000000.0
>>> type(.4e7)
<class 'float'>
>>> 4.2e-4
0.00042
```

Floating-Point Numbers

1. The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character **e** or **E** followed by a positive or negative integer may be appended to specify scientific notation:
2. The maximum value a floating-point number can have is approximately $1.8x10^{308}$. Python will indicate a number greater than that by the string inf:

Floating-Point Numbers

1. The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character **e** or **E** followed by a positive or negative integer may be appended to specify scientific notation:
2. The maximum value a floating-point number can have is approximately $1.8x10^{308}$. Python will indicate a number greater than that by the string inf:

Python >>>

```
>>> 1.79e308
1.79e+308
>>> 1.8e308
inf
```

Floating-Point Numbers

1. The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character **e** or **E** followed by a positive or negative integer may be appended to specify scientific notation:
2. The maximum value a floating-point number can have is approximately $1.8x10^{308}$. Python will indicate a number greater than that by the string inf:
3. The closest a nonzero number can be to zero is approximately $5.0X10^{-324}$. Anything closer to zero than that is effectively zero:

Floating-Point Numbers

1. The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character **e** or **E** followed by a positive or negative integer may be appended to specify scientific notation:
2. The maximum value a floating-point number can have is approximately $1.8x10^{308}$. Python will indicate a number greater than that by the string inf:
3. The closest a nonzero number can be to zero is approximately $5.0X10^{-324}$. Anything closer to zero than that is effectively zero:

Python >>>

```
>>> 5e-324
5e-324
>>> 1e-325
0.0
```

Complex Numbers

Complex numbers are specified as **<real part>+<imaginary part>j.**

Python

```
>>> 2+3j
(2+3j)
>>> type(2+3j)
<class 'complex'>
```

Boolean Type, Boolean Context, and "Truthiness".

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

1. $a == b$: a is equal to b,

Boolean Type, Boolean Context, and "Truthiness".

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

1. $a == b$: a is equal to b,
2. $a != b$: a is not equal to b,

Boolean Type, Boolean Context, and "Truthiness".

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

1. $a == b$: a is equal to b,
2. $a! = b$: a is not equal to b,
3. $a > b$: a is greater than b,

Boolean Type, Boolean Context, and "Truthiness".

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

1. $a == b$: a is equal to b,
2. $a! = b$: a is not equal to b,
3. $a > b$: a is greater than b,
4. $a < b$: a is less than b,

Boolean Type, Boolean Context, and "Truthiness".

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

1. $a == b$: a is equal to b,
2. $a! = b$: a is not equal to b,
3. $a > b$: a is greater than b,
4. $a < b$: a is less than b,
5. $a >= b$: a is greater than or equal to b,

Boolean Type, Boolean Context, and "Truthiness".

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

1. $a == b$: a is equal to b,
2. $a! = b$: a is not equal to b,
3. $a > b$: a is greater than b,
4. $a < b$: a is less than b,
5. $a >= b$: a is greater than or equal to b,
6. $a <= b$: a is less than or equal to b.

Boolean Type, Boolean Context, and "Truthiness".

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

1. $a == b$: a is equal to b,
2. $a! = b$: a is not equal to b,
3. $a > b$: a is greater than b,
4. $a < b$: a is less than b,
5. $a >= b$: a is greater than or equal to b,
6. $a <= b$: a is less than or equal to b.

Note: expressions $=>$ and $=<$ do not exist in Python.

Boolean Type, Boolean Context, and "Truthiness".

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

1. $a == b$: a is equal to b,
2. $a! = b$: a is not equal to b,
3. $a > b$: a is greater than b,
4. $a < b$: a is less than b,
5. $a >= b$: a is greater than or equal to b,
6. $a <= b$: a is less than or equal to b.

Note: expressions $=>$ and $=<$ do not exist in Python.

Try: $1 == 2$; $1 > 3$; $5! = 3$; $x = 2; x + 4 >= 1$.

IF TEST

Example

```
If a==b:
    print('The two numbers are equal')
else:
    print('The two numbers are different')
```

IF TEST

Example

```
If a==b:
    print('The two numbers are equal')
else:
    print('The two numbers are different')
```

Example

```
def testequal(a,b):
    If a==b:
        print('The two numbers are equal')
        return(2*a)
    else:
        print('The two numbers are different')
        return (a+b)
```

Outline

Session III

1. Recall+Exercise
2. Built-in functions
3. Operations
4. Conditional execution
5. Simple examples

Exercise

Write a function that returns the factorial of a number "n".

Exercise

Write a function that returns the factorial of a number "n".

Hint

$n! = (n-1)!.n \ if \ n \neq 0,$
$0! = 1$

Exercise

Write a function that returns the factorial of a number "n".

Hint

$n! = (n-1)!.n \; if \; n \neq 0,$
$0! = 1$

Solution

```
>>> def fact(n):
...     if n!=0:
...             return(fact(n-1)*n)
...     else:
...             return(1)
...
>>> fact(5)
120
>>>
```

Exercise

Write a function that returns the factorial of a number "n".

Hint

$n! = (n-1)!.n \text{ } if \text{ } n \neq 0,$
$0! = 1$

Solution

```
>>> def fact(n):
...     if n!=0:
...             return(fact(n-1)*n)
...     else:
...             return(1)
...
>>> fact(5)
120
>>>
```

What happens if we test fact(-2)? Find a solution. fact(0.5)?

Strings

1. Strings are sequences of character data. In Python, it is called str.
2. String literals are delimited using either single quote ′ ′ or double quotes " ". All the characters between the opening and closing delimiter are part of the string:

Strings

1. Strings are sequences of character data. In Python, it is called str.
2. String literals are delimited using either single quote ' ' or double quotes " ". All the characters between the opening and closing delimiter are part of the string:

Python

```
>>> print("I am a string.")
I am a string.
>>> type("I am a string.")
<class 'str'>

>>> print('I am too.')
I am too.
>>> type('I am too.')
<class 'str'>
```

Strings

1. Strings are sequences of character data. In Python, it is called str.
2. String literals are delimited using either single quote ' ' or double quotes " ". All the characters between the opening and closing delimiter are part of the string:
3. A string in Python can contain as many characters as you wish. It can also be empty:

Strings

1. Strings are sequences of character data. In Python, it is called str.
2. String literals are delimited using either single quote ' ' or double quotes " ". All the characters between the opening and closing delimiter are part of the string:
3. A string in Python can contain as many characters as you wish. It can also be empty:

Python

```
>>> ''
''
```

Strings

1. Strings are sequences of character data. In Python, it is called str.
2. String literals are delimited using either single quote ' ' or double quotes " ". All the characters between the opening and closing delimiter are part of the string:
3. A string in Python can contain as many characters as you wish. It can also be empty:
4. If you want to include a quote character as part of the string itself, you get:

Strings

1. Strings are sequences of character data. In Python, it is called str.
2. String literals are delimited using either single quote ' ' or double quotes " ". All the characters between the opening and closing delimiter are part of the string:
3. A string in Python can contain as many characters as you wish. It can also be empty:
4. If you want to include a quote character as part of the string itself, you get:

Python

```
>>> print('This string contains a single quote (') character.')
SyntaxError: invalid syntax
```

Strings

1. Strings are sequences of character data. In Python, it is called str.
2. String literals are delimited using either single quote ' ' or double quotes " ". All the characters between the opening and closing delimiter are part of the string:
3. A string in Python can contain as many characters as you wish. It can also be empty:
4. If you want to include a quote character as part of the string itself, you get: **To Solve:**

Python

```
>>> print("This string contains a single quote (') character.")
This string contains a single quote (') character.

>>> print('This string contains a double quote (") character.')
This string contains a double quote (") character.
```

Strings

1. Strings are sequences of character data. In Python, it is called str.
2. String literals are delimited using either single quote ' ' or double quotes " ". All the characters between the opening and closing delimiter are part of the string:
3. A string in Python can contain as many characters as you wish. It can also be empty:
4. If you want to include a quote character as part of the string itself, you get:
   To Solve:
   OR

Python

```
>>> print('This string contains a single quote (\') character.')
This string contains a single quote (') character.
```

Strings

1. Strings are sequences of character data. In Python, it is called str.
2. String literals are delimited using either single quote ' ' or double quotes " ". All the characters between the opening and closing delimiter are part of the string:
3. A string in Python can contain as many characters as you wish. It can also be empty:
4. If you want to include a quote character as part of the string itself, you get:
   To Solve:
   OR

Python

```
>>> print("This string contains a double quote (\") character.")
This string contains a double quote (") character.
```

Escape sequences

Escape Sequence	Usual Interpretation of Character(s) After Backslash	"Escaped" Interpretation
`\'`	Terminates string with single quote opening delimiter	Literal single quote (') character
`\"`	Terminates string with double quote opening delimiter	Literal double quote (") character
`\newline`	Terminates input line	Newline is ignored
`\\`	Introduces escape sequence	Literal backslash (\) character

2.Built-in functions

Print()-function

- Pressing **Enter** in the middle of a string will cause Python to think it is incomplete:

```
Python
>>> print('a

SyntaxError: EOL while scanning string literal
```

Print()-function

- Pressing **Enter** in the middle of a string will cause Python to think it is incomplete:
- To break up a string over more than one line:

Python

```
>>> print('a\
... b\
... c')
abc
```

Print()-function

- Pressing **Enter** in the middle of a string will cause Python to think it is incomplete:
- To break up a string over more than one line:
- To include a literal backslash in a string:

Python

```
>>> print('foo\\bar')
foo\bar
```

Print()-function

- Pressing **Enter** in the middle of a string will cause Python to think it is incomplete:
- To break up a string over more than one line:
- To include a literal backslash in a string:
- A tab character can be specified by the escape sequence "\t"

Python

```
>>> print('foo\tbar')
foo     bar
```

Print()-function

- Pressing **Enter** in the middle of a string will cause Python to think it is incomplete:
- To break up a string over more than one line:
- To include a literal backslash in a string:
- A tab character can be specified by the escape sequence "\t"
- Some examples

```
Python
>>> print("a\tb")
a    b
>>> print("a\141\x61")
aaa
>>> print("a\nb")
a
b
>>> print('\u2192 \N{rightwards arrow}')
→ →
```

Print()-function

- Pressing **Enter** in the middle of a string will cause Python to think it is incomplete:
- To break up a string over more than one line:
- To include a literal backslash in a string:
- A tab character can be specified by the escape sequence "\t"
- Some examples
- A **raw string** literal is preceded by r or R , which specifies that escape sequences in the associated string are not translated.

```
Python
>>> print('foo\nbar')
foo
bar
>>> print(r'foo\nbar')
foo\nbar

>>> print('foo\\bar')
foo\bar
>>> print(R'foo\\bar')
foo\\bar
```

Print()-function

- Pressing **Enter** in the middle of a string will cause Python to think it is incomplete:
- To break up a string over more than one line:
- To include a literal backslash in a string:
- A tab character can be specified by the escape sequence $"\backslash t"$
- Some examples
- A **raw string** literal is preceded by r or R , which specifies that escape sequences in the associated string are not translated.
- Triple-Quoted Strings provides a convenient way to create a string with both single and double quotes in it.

Python

```
>>> print('''This string has a single (') and a double (") quote.''')
This string has a single (') and a double (") quote.
```

Built-in functions

Python >>>

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Built-in functions

Math

Function	Description
`abs()`	Returns absolute value of a number
`divmod()`	Returns quotient and remainder of integer division
`max()`	Returns the largest of the given arguments or items in an iterable
`min()`	Returns the smallest of the given arguments or items in an iterable
`pow()`	Raises a number to a power
`round()`	Rounds a floating-point value
`sum()`	Sums the items of an iterable

Conversion of types

The duty of the functions *int*(), *float*(),and *str*() is to convert the type.
To pass from integer to string or float and vice-versa.

Conversion of types

The duty of the functions *int*(), *float*(),and *str*() is to convert the type.
To pass from integer to string or float and vice-versa.

```
lama@lama-pc: ~
File  Edit  View  Search  Terminal  Help
(base) lama@lama-pc:~$ python3
Python 3.8.3 (default, Jul  2 2020, 16:21:59)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> i=3
>>> str(i)
'3'
>>> i='456'
>>> int(i)
456
>>> float(i)
456.0
>>> i='3.1416'
>>> float(i)
3.1416
>>> 
```

3.Operations

Numerical operations

Using Python shell as a calculator : introducing in Python basic mathematical operations.

1. **A-S** Addiction $a + b$ and Subtraction $a - b$;

Numerical operations

Using Python shell as a calculator : introducing in Python basic mathematical operations.

1. **A-S** Addiction $a + b$ and Subtraction $a - b$;
2. **M-D** Multiplication $a * b$ and Division a/b;

Numerical operations

Using Python shell as a calculator : introducing in Python basic mathematical operations.

1. **A-S** Addiction $a + b$ and Subtraction $a - b$;
2. **M-D** Multiplication $a * b$ and Division a/b;
3. **E** Exponential $a * *b$;

Numerical operations

Using Python shell as a calculator : introducing in Python basic mathematical operations.

1. **A-S** Addiction $a + b$ and Subtraction $a - b$;
2. **M-D** Multiplication $a * b$ and Division a/b;
3. **E** Exponential $a * *b$;
4. **P** Parenthesis(...), are used to force the order of operations;

Numerical operations

Using Python shell as a calculator : introducing in Python basic mathematical operations.

1. **A-S** Addiction $a + b$ and Subtraction $a - b$;
2. **M-D** Multiplication $a * b$ and Division a/b;
3. **E** Exponential $a * *b$;
4. **P** Parenthesis(...), are used to force the order of operations;

Left-to-right order: remember the acronymous "PEMDAS":

$$P > E > M \sim D > A \sim S.$$

Numerical operations

Using Python shell as a calculator : introducing in Python basic mathematical operations.

1. **A-S** Addiction $a + b$ and Subtraction $a - b$;
2. **M-D** Multiplication $a * b$ and Division a/b;
3. **E** Exponential $a * *b$;
4. **P** Parenthesis(...), are used to force the order of operations;

Left-to-right order: remember the acronymous "PEMDAS":

$$P > E > M \sim D > A \sim S.$$

5- **F** Floor division $a//b$ $(= \lfloor a/b \rfloor, integerpart)$;

Numerical operations

Using Python shell as a calculator : introducing in Python basic mathematical operations.

1. **A-S** Addiction $a + b$ and Subtraction $a - b$;
2. **M-D** Multiplication $a * b$ and Division a/b;
3. **E** Exponential $a * *b$;
4. **P** Parenthesis(...), are used to force the order of operations;

Left-to-right order: remember the acronymous "PEMDAS":

$$P > E > M \sim D > A \sim S.$$

5- **F** Floor division $a//b$ $(= \lfloor a/b \rfloor, integerpart)$;

5- **R** Rest $a\%b$ $(= a - b * a//b)$.

Numerical operations

Using Python shell as a calculator : introducing in Python basic mathematical operations.

1. **A-S** Addiction $a + b$ and Subtraction $a - b$;
2. **M-D** Multiplication $a * b$ and Division a/b;
3. **E** Exponential $a * *b$;
4. **P** Parenthesis(...), are used to force the order of operations;

Left-to-right order: remember the acronymous "PEMDAS":

$$P > E > M \sim D > A \sim S.$$

5- **F** Floor division $a//b$ $(= \lfloor a/b \rfloor, integerpart)$;

5- **R** Rest $a\%b$ $(= a - b * a//b)$.

Left-to-right order: $F \sim R \sim M \sim D$.

Example

Predict the answer in Python: $2 + 3$; $2 * (3 - 1)$; $2 * 3 - 1$; $(1 + 1) * *(5 - 2)$; $3 * 1 * *3$; $2/3$; $2/0$; $(3 * 1) * *3$; $4//3$; $2//3$; $17\%3$ and $15\%4$.

Example

Predict the answer in Python: $2 + 3$; $2 * (3 - 1)$; $2 * 3 - 1$; $(1 + 1) * *(5 - 2)$; $3 * 1 * *3$; $2/3$; $2/0$; $(3 * 1) * *3$; $4//3$; $2//3$; 17%3 and 15%4.

```
>>> 2+3
5
```
...
```
>>> 2*3-1
5
```

```
>>> 3*1**3
3
```
...
```
>>> (3*1)**3
27
```
```
>>> 4//3
1
```
...
```
>>> 2//3
0
```

```
>>> 2*(3-1)
4
```
...
```
>>> (1+1)**(5-2)
8
```
```
>>> 2/3
0.6666666666666666
```
..
```
>>> 2/0
Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
2/0
ZeroDivisionError:  int division or modu
```
```
>>> 17%3
2
```
...
```
>>> 15%4
3
```

Operations on Strings

1. $*$ is used for the **repetition**;

Operations on Strings

1. * is used for the **repetition**;
2. + is used for the **concatenation**;

Operations on Strings

1. * is used for the **repetition**;
2. + is used for the **concatenation**;

```
>>> 'hello'*3
'hellohellohello'
```

...

```
>>> 3*'hello'
'hellohellohello'
```

...

```
>>> 'hello' + 'world'
'helloworld'
```

Operations on Strings

1. * is used for the **repetition**;
2. + is used for the **concatenation**;

```
>>> 'hello'*3
'hellohellohello'
```
...
```
>>> 3*'hello'
'hellohellohello'
```
...
```
>>> 'hello' + 'world'
'helloworld'
```

3. # is used to **insert comments** in the text.

```
>>> 'hello' + 'world' # concatenation of strings
'helloworld'
```

Exercise I

Predict the answer of the following expressions, then execute it on Python.

- `(1+2)**3`
- `"Da" * 4`
- `"Da" + 3`
- `("Pa"+"La") * 2`
- `("Da"*4) / 2`
- `5 / 2`

- `5 // 2`
- `5 % 2`

- `str(4) * int("3")`
- `int("3") + float("3.2")`
- `str(3) * float("3.2")`
- `str(3/4) * 2`

Exercise II-III

Exercise 2: **What is the volume of a sphere with radius 5?**

Exercise II-III

Exercise 2: **What is the volume of a sphere with radius 5?**

```
r = 5 #sphere radious
print(4/3 *3.14 *(r**3)) #shpere volume
```

```
>>>
523.333333333
```

Exercise II-III

Exercise 2: **What is the volume of a sphere with radius 5?**

```
r = 5 #sphere radious
print(4/3 *3.14 *(r**3)) #shpere volume
```

```
>>>
523.333333333
```

Exercise 3: **Suppose the cover price of a book is** 24.95 **$, but book stores get a 40% discount. Shipping costs** 3**$ for the first copy and** 75 **cents for each additional copy. What is the total wholesale cost for** 60 **copies?**

Exercise II-III

Exercise 2: **What is the volume of a sphere with radius 5?**

```
r = 5 #sphere radious
print(4/3 *3.14 *(r**3)) #shpere volume
```

```
>>>
523.333333333
```

Exercise 3: **Suppose the cover price of a book is** 24.95 **$, but book stores get a 40% discount. Shipping costs** 3**$ for the first copy and** 75 **cents for each additional copy. What is the total wholesale cost for** 60 **copies?**

```
p = 24.95 #original price
s = p*60/100 #discount price
t1 = s+3 #first book delivered
t2 = s+0.75 #other books delivered
print(t1 + 59*t2) #final price
```

```
>>>
945.45
```

Outline

Session IV

1. Modulus
2. Lists
3. Operations on Lists

1.Modulus

Modulus

Module: is a file that contains a collection of functions.

Modulus

Module: is a file that contains a collection of functions.

If we run the name of the modulus we can learn if it is already in our version of Python.

Modulus

Module: is a file that contains a collection of functions.

If we run the name of the modulus we can learn if it is already in our version of Python.

To use one of the function in the modulus:

Modulus

Module: is a file that contains a collection of functions.

If we run the name of the modulus we can learn if it is already in our version of Python.

To use one of the function in the modulus:

1. Open the module with command import;

Modulus

Module: is a file that contains a collection of functions.

If we run the name of the modulus we can learn if it is already in our version of Python.

To use one of the function in the modulus:

1. Open the module with command import;
2. Specify the name of the module and the name of the function, separated by a dot (dot notation).

Math Module

```
>>> math
<module 'math' (built-in)>
```

...

```
>>> import math
>>>
```

...

```
>>> import math
>>> math.pi
3.141592653589793
```

...

```
>>> import math
>>> math.e
2.718281828459045
```

Math Module

```
>>> math
<module 'math' (built-in)>
```

...

```
>>> import math
>>>
```

...

```
>>> import math
>>> math.pi
3.141592653589793
```

...

```
>>> import math
>>> math.e
2.718281828459045
```

Module name can be reassigned using command as.

Math Module

```
>>> math
<module 'math' (built-in)>
```

...

```
>>> import math
>>>
```

...

```
>>> import math
>>> math.pi
3.141592653589793
```

...

```
>>> import math
>>> math.e
2.718281828459045
```

Module name can be reassigned using command as.

```
>>> import math as mt
>>> mt.e
2.718281828459045
```

...

```
>>> mt.sin(0)
>>> 0.0
```

...

```
>>> mt.sin(90)
>>> 0.8939966636005579
```

```
>>> degrees = 90
>>> radians = degrees / 360.0 * 2 * math.pi
>>> math.sin(rad)
1.0
```

Exercises

1-Calculate the sine of the angle $\frac{3\pi}{4}$.

Exercises

1-Calculate the sine of the angle $\frac{3\pi}{4}$.

```
lama@lama-pc: ~
File  Edit  View  Search  Terminal  Help
(base) lama@lama-pc:~$ python3
Python 3.8.3 (default, Jul  2 2020, 16:21:59)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> math.sin(3*math.pi/4)
0.7071067811865476
>>> 
```

2-Calculate the length of the hypotenuse of a right triangle having 9 and 12 as dimensions for the other two sides.

Exercises

1-Calculate the sine of the angle $\frac{3\pi}{4}$.

```
lama@lama-pc: ~
File  Edit  View  Search  Terminal  Help
(base) lama@lama-pc:~$ python3
Python 3.8.3 (default, Jul  2 2020, 16:21:59)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> math.sin(3*math.pi/4)
0.7071067811865476
>>>
```

2-Calculate the length of the hypotenuse of a right triangle having 9 and 12 as dimensions for the other two sides.

```
>>> def hypo(a,b):
...     h=a*a+b*b
...     s=math.sqrt(h)
...     return(s)
...
>>> hypo(3,4)
5.0
>>> hypo(9,12)
15.0
```

Random Module

- random.**random()**
  Return the next random floating point number in the range $[0.0, 1.0)$.

Random Module

- random.**random()**
  Return the next random floating point number in the range $[0.0, 1.0)$.
- random.**uniform(a, b)**
  Return a random floating point number N such that $a <= N <= b$ for $a <= b$ and $b <= N <= a$ for $b < a$.

Random Module

- random.**random()**
  Return the next random floating point number in the range $[0.0, 1.0)$.
- random.**uniform(a, b)**
  Return a random floating point number N such that $a <= N <= b$ for $a <= b$ and $b <= N <= a$ for $b < a$.
- random.**randrange(stop)** or random.**randrange(start, stop[, step])**
  Return a randomly selected element from range(start, stop, step).

Random Module

- random.**random()**
  Return the next random floating point number in the range $[0.0, 1.0)$.
- random.**uniform(a, b)**
  Return a random floating point number N such that $a <= N <= b$ for $a <= b$ and $b <= N <= a$ for $b < a$.
- random.**randrange(stop)** or random.**randrange(start, stop[, step])**
  Return a randomly selected element from range(start, stop, step).
- random.**randint(a, b)**
  Return a random integer N such that $a <= N <= b$.

Random Module

- random.**random()**
  Return the next random floating point number in the range $[0.0, 1.0)$.
- random.**uniform(a, b)**
  Return a random floating point number N such that $a <= N <= b$ for $a <= b$ and $b <= N <= a$ for $b < a$.
- random.**randrange(stop)** or random.**randrange(start, stop[, step])**
  Return a randomly selected element from range(start, stop, step).
- random.**randint(a, b)**
  Return a random integer N such that $a <= N <= b$.
- random.**choice(seq)**
  Return a random element from the non-empty sequence seq. If seq is empty, raises IndexError.

Random Module

- random.**random()**
  Return the next random floating point number in the range $[0.0, 1.0)$.
- random.**uniform(a, b)**
  Return a random floating point number N such that $a <= N <= b$ for $a <= b$ and $b <= N <= a$ for $b < a$.
- random.**randrange(stop)** or random.**randrange(start, stop[, step])**
  Return a randomly selected element from range(start, stop, step).
- random.**randint(a, b)**
  Return a random integer N such that $a <= N <= b$.
- random.**choice(seq)**
  Return a random element from the non-empty sequence seq. If seq is empty, raises IndexError.
- random.**shuffle(seq)**
  Returns a shuffeled sequence.

Random Module

- random.**random()**
  Return the next random floating point number in the range $[0.0, 1.0)$.
- random.**uniform(a, b)**
  Return a random floating point number N such that $a <= N <= b$ for $a <= b$ and $b <= N <= a$ for $b < a$.
- random.**randrange(stop)** or random.**randrange(start, stop[, step])**
  Return a randomly selected element from range(start, stop, step).
- random.**randint(a, b)**
  Return a random integer N such that $a <= N <= b$.
- random.**choice(seq)**
  Return a random element from the non-empty sequence seq. If seq is empty, raises IndexError.
- random.**shuffle(seq)**
  Returns a shuffeled sequence.
- random.**sample(seq, k)**
  Return a k length list of unique elements chosen from the sequence.

Examples

```
>>> random()                              # Random float:  0.0 <= x < 1.0
0.37444887175646646

>>> uniform(2.5, 10.0)                    # Random float:  2.5 <= x < 10.0
3.1800146073117523

>>> expovariate(1 / 5)                    # Interval between arrivals averagi
5.148957571865031

>>> randrange(10)                         # Integer from 0 to 9 inclusive
7

>>> randrange(0, 101, 2)                  # Even integer from 0 to 100 inclus
26

>>> choice(['win', 'lose', 'draw'])       # Single random element from a sequ
'draw'

>>> deck = 'ace two three four'.split()
>>> shuffle(deck)                         # Shuffle a list
>>> deck
['four', 'two', 'ace', 'three']

>>> sample([10, 20, 30, 40, 50], k=4)    # Four samples without replacement
```

Exercise

Write a Python programm allowing us to get the roots of a second degree equation when they exist.

Exercise

Write a Python programm allowing us to get the roots of a second degree equation when they exist.

```
In [5]: import math
        def roots(a,b,c):
            d=b**2-4*a*c
            if d>0:
                r=math.sqrt(d)
                x1=(-b-r)/(2*a)
                x2=(-b+r)/(2*a)
                return('There exists two distinct solutions:',x1,x2)
            if d==0:
                r=math.sqrt(d)
                x=-b/(2*a)
                return('There exists one double solution:',x)
            else:
                return('the equation has no solution in R2')

In [6]: roots(1,2,1)

Out[6]: ('There exists one double solution:', -1.0)

In [7]: roots(1,-4,5)

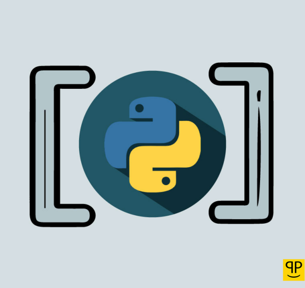
Out[7]: 'the equation has no solution in R2'

In [8]: roots(1,5,6)

Out[8]: ('There exists two distinct solutions:', -3.0, -2.0)
```

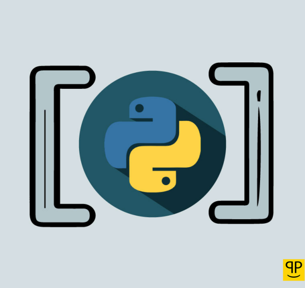
2.Lists

Lists in Python



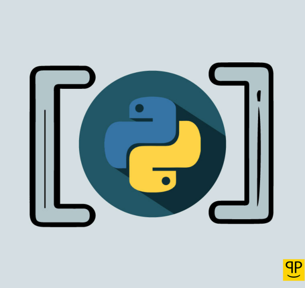
1. A list is a **data structure** in Python that is a **mutable**, or **changeable, ordered sequence** of elements.

Lists in Python



1. A list is a **data structure** in Python that is a **mutable**, or **changeable, ordered sequence** of elements.
2. Each element or value that is inside of a list is called an **item**.

Lists in Python



1. A list is a **data structure** in Python that is a **mutable**, or **changeable, ordered sequence** of elements.
2. Each element or value that is inside of a list is called an **item**.
3. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ].

4. Lists are great to use when you want to work with many **related values**.

4. Lists are great to use when you want to work with many **related values**.
5. They enable you to keep data together that belongs together, condense your code, and perform the same methods and operations on multiple values at once.

4. Lists are great to use when you want to work with many **related values**.

5. They enable you to keep data together that belongs together, condense your code, and perform the same methods and operations on multiple values at once.

6. When thinking about Python lists and other data structures that are types of collections, it is useful to consider all the different collections you have on your computer: your assortment of files, your song playlists, your browser bookmarks, your emails, the collection of videos you can access on a streaming service, and more.

To get started, let's create a list that contains items of the string data type:

```
sea_creatures = ['shark', 'cuttlefish', 'squid', 'mantis shrimp', 'anemone']
```

To get started, let's create a list that contains items of the string data type:

```
sea_creatures = ['shark', 'cuttlefish', 'squid', 'mantis shrimp', 'anemone']
```

When we print out the list, the output looks exactly like the list we created:

```
print(sea_creatures)
```

```
Output
['shark', 'cuttlefish', 'squid', 'mantis shrimp', 'anemone']
```

To get started, let's create a list that contains items of the string data type:

```
sea_creatures = ['shark', 'cuttlefish', 'squid', 'mantis shrimp', 'anemone']
```

When we print out the list, the output looks exactly like the list we created:

```
print(sea_creatures)
```

```
Output
['shark', 'cuttlefish', 'squid', 'mantis shrimp', 'anemone']
```

As an ordered sequence of elements, each item in a list can be called individually, through **indexing**.

Indexing lists

Each item in a list corresponds to an index number, which is an integer value, starting with the index number 0.

Indexing lists

Each item in a list corresponds to an index number, which is an integer value, starting with the index number 0.

'shark'	'cuttlefish'	'squid'	'mantis shrimp'	'anemone'
0	1	2	3	4

Indexing lists

Each item in a list corresponds to an index number, which is an integer value, starting with the index number 0.

'shark'	'cuttlefish'	'squid'	'mantis shrimp'	'anemone'
0	1	2	3	4

Because each item in a Python list has a corresponding index number, we're able to access and manipulate lists in the same ways we can with other sequential data types.

Indexing lists

Each item in a list corresponds to an index number, which is an integer value, starting with the index number 0.

'shark'	'cuttlefish'	'squid'	'mantis shrimp'	'anemone'
0	1	2	3	4

Because each item in a Python list has a corresponding index number, we're able to access and manipulate lists in the same ways we can with other sequential data types.

```
sea_creatures[0] = 'shark'
sea_creatures[1] = 'cuttlefish'
sea_creatures[2] = 'squid'
sea_creatures[3] = 'mantis shrimp'
sea_creatures[4] = 'anemone'
```

Negative indexing

NB: If we call the list with an index numberthat is greater than 4, it will be out of range as it will not be valid:

Negative indexing

NB: If we call the list with an index numberthat is greater than 4, it will be out of range as it will not be valid:

```
print(sea_creatures[18])
```

```
Output

IndexError: list index out of range
```

Negative indexing

NB: If we call the list with an index numberthat is greater than 4, it will be out of range as it will not be valid:

```
print(sea_creatures[18])
```

```
Output

IndexError: list index out of range
```

In addition to positive index numbers, we can also access items from the list with a negative index number, by counting backwards from the end of the list, starting at -1.

Negative indexing

NB: If we call the list with an index numberthat is greater than 4, it will be out of range as it will not be valid:

```
print(sea_creatures[18])
```

```
Output
IndexError: list index out of range
```

In addition to positive index numbers, we can also access items from the list with a negative index number, by counting backwards from the end of the list, starting at -1.

'shark'	'cuttlefish'	'squid'	'mantis shrimp'	'anemone'
-5	-4	-3	-2	-1

Negative indexing

NB: If we call the list with an index numberthat is greater than 4, it will be out of range as it will not be valid:

```
print(sea_creatures[18])
```

```
Output
IndexError: list index out of range
```

In addition to positive index numbers, we can also access items from the list with a negative index number, by counting backwards from the end of the list, starting at -1.

```
print(sea_creatures[-3])
```

```
Output
squid
```

Modifying Lists

If we want to change the string value of the item at index 1 from 'cuttlefish' to 'octopus', we can do so like this:

Modifying Lists

If we want to change the string value of the item at index 1 from 'cuttlefish' to 'octopus', we can do so like this:

```
sea_creatures[1] = 'octopus'
```

Now when we print `sea_creatures`, the list will be different:

```
print(sea_creatures)
```

```
Output
['shark', 'octopus', 'squid', 'mantis shrimp', 'anemone']
```

Modifying Lists

If we want to change the string value of the item at index 1 from 'cuttlefish' to 'octopus', we can do so like this:
We can also change the value of an item by using a **negative index number** instead:

Modifying Lists

If we want to change the string value of the item at index 1 from 'cuttlefish' to 'octopus', we can do so like this:
We can also change the value of an item by using a **negative index number** instead:

```
sea_creatures[-3] = 'blobfish'
print(sea_creatures)
```

```
Output
['shark', 'octopus', 'blobfish', 'mantis shrimp', 'anemone']
```

Slicing lists

We can also call out a few items from the list.
Let's say we would like to just print the middle items of the list, we can do so by creating a slice. With slices, we can call multiple values by creating a range of index numbers separated by a colon $[x : y]$:

Slicing lists

We can also call out a few items from the list.
Let's say we would like to just print the middle items of the list, we can do so by creating a slice. With slices, we can call multiple values by creating a range of index numbers separated by a colon $[x : y]$:

```
print(sea_creatures[1:4])
```

```
Output
['octopus', 'blobfish', 'mantis shrimp']
```

Slicing lists

We can also call out a few items from the list.
Let's say we would like to just print the first items of the list till n^{th} item of the list, we can do so by creating a slice. With slices, we can call multiple values by creating a range of index numbers separated by a colon [: n]:

```
print(sea_creatures[:3])
```

```
Output
['shark', 'octopus', 'blobfish']
```

Slicing lists

We can also call out a few items from the list.
Let's say we would like to just print the items from n till the end of the list, we can do so by creating a slice. With slices, we can call multiple values by creating a range of index numbers separated by a colon $[n:]$:

```
print(sea_creatures[2:])
```

```
Output
['blobfish', 'mantis shrimp', 'anemone']
```

Slicing lists

We can also call out a few items from the list.
Let's say we would like to just print the middle items with negative index of the list, we can do so by creating a slice. With slices, we can call multiple values by creating a range of index numbers separated by a colon $[-x:-y]$:

```
print(sea_creatures[-4:-2])
print(sea_creatures[-3:])
```

```
Output
['octopus', 'blobfish']
['blobfish', 'mantis shrimp', 'anemone']
```

One last parameter that we can use with slicing is called **stride** , which refers to how many items to move forward after the first item is retrieved from the list.

One last parameter that we can use with slicing is called **stride** , which refers to how many items to move forward after the first item is retrieved from the list. The syntax for this construction is list $[x : y : z]$, with z referring to stride.

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

print(numbers[1:11:2])
```

```
Output
[1, 3, 5, 7, 9]
```

One last parameter that we can use with slicing is called **stride** , which refers to how many items to move forward after the first item is retrieved from the list. We can omit the first two parameters and use stride alone as a parameter with the syntax *list*[:: *z*]:

```
print(numbers[::3])
```

```
Output
[0, 3, 6, 9, 12]
```

1.Operations on Lists

Operations on Lists

1) Adding two lists using + to concatenate them:

$l_1 + l_2 = l_3$

```
sea_creatures = ['shark', 'octopus', 'blobfish', 'mantis shrimp', 'anemone']
oceans = ['Pacific', 'Atlantic', 'Indian', 'Southern', 'Arctic']

print(sea_creatures + oceans)
```

```
['shark', 'octopus', 'blobfish', 'mantis shrimp', 'anemone', 'Pacific', 'Atlantic', 'Indian', 'Southern', 'Arctic']
```

Operations on Lists

1) **Adding two lists using + to concatenate them:**
2) **Extending a list with another list using *extend*():**

$l_1.extend(l_2) = l_1;\ l_1 = l_1 + l_2$

```
In [9]:  l1=[1,2,3,4,5,6]
         l2=[11,12]

In [10]: l1.extend(l2)

In [11]: l1

Out[11]: [1, 2, 3, 4, 5, 6, 11, 12]
```

Operations on Lists

1) **Adding two lists using + to concatenate them:**
2) **Extending a list with another list using *extend*():**
3) **Adding an item to the list using *append*():**

$l_1.append(a) = l_1;$

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
```

Deletion

We must distinguish between two ideas, revoming and deleting :

Deletion

We must distinguish between two ideas, revoming and deleting :
A specific item *a* can be removed from a list "*l*" as follows:

Deletion

We must distinguish between two ideas, revoming and deleting :
A specific item *a* can be removed from a list "*l*" as follows:

```
In [12]: l1
Out[12]: [1, 2, 3, 4, 5, 6, 11, 12]

In [13]: l1.remove(5)

In [14]: l1
Out[14]: [1, 2, 3, 4, 6, 11, 12]
```

Deletion

We must distinguish between two ideas, revoming and deleting :
A specific item *a* can be removed from a list "*l*" as follows:
An item at position *n* can be deleted from a list "*l*" as follows:

Deletion

We must distinguish between two ideas, revoming and deleting :
A specific item *a* can be removed from a list "*l*" as follows:
An item at position *n* can be deleted from a list "*l*" as follows:

```
# list of numbers
n_list = [1, 2, 3, 4, 5, 6]

# Deleting 2nd element
del n_list[1]
```

Deletion

We must distinguish between two ideas, revoming and deleting :
A specific item a can be removed from a list "l" as follows:
An item at position n can be deleted from a list "l" as follows:
We can remove all the items from the list by using:

Deletion

We must distinguish between two ideas, revoming and deleting :
A specific item *a* can be removed from a list "*l*" as follows:
An item at position *n* can be deleted from a list "*l*" as follows:
We can remove all the items from the list by using:

```
In [16]: l1
Out[16]: [1, 2, 3, 6, 11, 12]

In [17]: l1.clear()

In [18]: l1
Out[18]: []
```

Deletion

We must distinguish between two ideas, revoming and deleting :
A specific item *a* can be removed from a list "*l*" as follows:
An item at position *n* can be deleted from a list "*l*" as follows:
We can remove all the items from the list by using:We can POP the last element of the list as follows:

Deletion

We must distinguish between two ideas, revoming and deleting :
A specific item *a* can be removed from a list "*l*" as follows:
An item at position *n* can be deleted from a list "*l*" as follows:
We can remove all the items from the list by using:We can POP the last element of the list as follows:

```
In [19]: l1=[1, 2, 3, 4, 5, 6, 11, 12]

In [20]: l1.pop()

Out[20]: 12
```

More operations

We can ask for the length of a list l by using:

More operations

We can ask for the length of a list *l* by using:

```
In [24]: l1
Out[24]: [1, 2, 3, 4, 5, 6, 11]

In [25]: len(l1)
Out[25]: 7
```

More operations

We can ask for the length of a list l by using:
We can count the occurences of a certain element a in l by using:

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:

```
In [29]: l2

Out[29]: [3, 2, 4, 3, 3, 2, 4, 5]

In [30]: l2.count(3)

Out[30]: 3
```

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:

```
In [32]: l1

Out[32]: [1, 2, 3, 4, 5, 6, 11]

In [33]: l1.index(3)

Out[33]: 2
```

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:
Note that: If a list contains the same element several times, we can ask for the index starting a certain position :

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:
Note that: If a list contains the same element several times, we can ask for the index starting a certain position :

```
In [50]: l2=[3,2,4,3,3,2,4,5]

In [51]: l2.index(3,2)

Out[51]: 3
```

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:
Note that: If a list contains the same element several times, we can ask for the index starting a certain position :
We can ask for the reverse of a list *l* by using:

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:
Note that: If a list contains the same element several times, we can ask for the index starting a certain position :
We can ask for the reverse of a list *l* by using:

```
In [38]: l1

Out[38]: [1, 2, 3, 4, 5, 6, 11]

In [39]: l1.reverse()

In [40]: l1

Out[40]: [11, 6, 5, 4, 3, 2, 1]
```

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:
Note that: If a list contains the same element several times, we can ask for the index starting a certain position :
We can ask for the reverse of a list *l* by using:
We can get a copy of the list *l* by using:

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:
Note that: If a list contains the same element several times, we can ask for the index starting a certain position :
We can ask for the reverse of a list *l* by using:
We can get a copy of the list *l* by using:

```
In [42]: l1
Out[42]: [11, 6, 5, 4, 3, 2, 1]

In [43]: l2=l1.copy()

In [44]: l2
Out[44]: [11, 6, 5, 4, 3, 2, 1]
```

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:
Note that: If a list contains the same element several times, we can ask for the index starting a certain position :
We can ask for the reverse of a list *l* by using:
We can get a copy of the list *l* by using:
We can sort a list *l* in a **decreasing or increasing** order as follows: (PS, by default it is an increasing order)

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:
Note that: If a list contains the same element several times, we can ask for the index starting a certain position :
We can ask for the reverse of a list *l* by using:
We can get a copy of the list *l* by using:
We can sort a list *l* in a **decreasing or increasing** order as follows: (PS, by default it is an increasing order)

```
In [44]: l2
Out[44]: [11, 6, 5, 4, 3, 2, 1]

In [45]: l2.sort()

In [46]: l2
Out[46]: [1, 2, 3, 4, 5, 6, 11]
```

More operations

We can ask for the length of a list *l* by using:
We can count the occurences of a certain element *a* in *l* by using:
We can ask for the index of an item in a list *l* by using:
Note that: If a list contains the same element several times, we can ask for the index starting a certain position :
We can ask for the reverse of a list *l* by using:
We can get a copy of the list *l* by using:
We can sort a list *l* in a **decreasing or increasing** order as follows: (PS, by default it is an increasing order)

```
In [46]: l2
Out[46]: [1, 2, 3, 4, 5, 6, 11]

In [48]: l2.sort(reverse=True)

In [49]: l2
Out[49]: [11, 6, 5, 4, 3, 2, 1]
```

Exercise

Using the operations on lists to write a function that returns the min of a list l.

Exercise

Using the operations on lists to write a function that returns the min of a list l.

```
In [58]: def minlist(l):
             l.sort()
             l.reverse()
             a=l.pop()
             return a
```

```
In [53]: l=[4,6,1,7,8,4,3,9,0]
```

```
In [59]: minlist(l)

Out[59]: 0
```

Exercise 2

Define the following list: $L = [17, 38, 10, 25, 72]$ then do the following:

1. sort and show the list.
2. add number 12 to the list.
3. Give the reverse of the list.
4. Give the index of the elemnt 17.
5. Remove the element 38 from the list.
6. Show the sub-list Q composed of the second to the third element of L.
7. Show the sub-list R composed of the elements of L from the beginning to the second element.
8. Show the sub-list T composed of the elements of L from its third element till the end.
9. Show the sub-list C composed of all the elements of L.
10. Show the last and the third element of L using negative indexing.

Solutions

```
In [1]: L=[17,38,10,25,72]

In [2]: L.sort()

In [3]: L
Out[3]: [10, 17, 25, 38, 72]

In [4]: L.append(12)

In [5]: L
Out[5]: [10, 17, 25, 38, 72, 12]

In [6]: L.reverse()

In [7]: L
Out[7]: [12, 72, 38, 25, 17, 10]

In [8]: L.index(17)
Out[8]: 4

In [9]: L.remove(38)

In [10]: L
Out[10]: [12, 72, 25, 17, 10]

In [14]: Q=L[1:3]

In [15]: Q
Out[15]: [72, 25]

In [16]: R=L[:2]

In [17]: R
Out[17]: [12, 72]
```

Solutions

```
In [18]: T=L[2:]

In [19]: T
Out[19]: [25, 17, 10]

In [20]: C=L.copy()

In [21]: C
Out[21]: [12, 72, 25, 17, 10]

In [22]: L[-1]
         l=len(L)
         L[-l+2]
Out[22]: 25

In [23]: l
Out[23]: 5

In [24]: L[-l+2]
Out[24]: 25

In [25]: L[-1]
Out[25]: 10
```

Range() and List() Functions

The instruction *range()* is a special function in **Python** that generates integer numbers included in a certain interval.

Range() and List() Functions

The instruction *range*() is a special function in **Python** that generates integer numbers included in a certain interval.
When it is combined with the *list*() function, it gives a list of integers.

Range() and List() Functions

The instruction *range*() is a special function in **Python** that generates integer numbers included in a certain interval.
When it is combined with the *list*() function, it gives a list of integers.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Range() and List() Functions

The instruction *range*() is a special function in **Python** that generates integer numbers included in a certain interval.
When it is combined with the *list*() function, it gives a list of integers.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

It is possible to give two or three arguments to the list function as shown:

Range() and List() Functions

The instruction *range*() is a special function in **Python** that generates integer numbers included in a certain interval.
When it is combined with the *list*() function, it gives a list of integers.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

It is possible to give two or three arguments to the list function as shown:

```
>>> list(range(0, 5))
[0, 1, 2, 3, 4]
>>> list(range(15, 20))
[15, 16, 17, 18, 19]
>>> list(range(0, 1000, 200))
[0, 200, 400, 600, 800]
>>> list(range(2, -2, -1))
[2, 1, 0, -1]
```

Range() and List() Functions

The instruction *range*() is a special function in **Python** that generates integer numbers included in a certain interval.
When it is combined with the *list*() function, it gives a list of integers.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

It is possible to give two or three arguments to the list function as shown:

```
>>> list(range(10,0))
[]
```

Range() and List() Functions

The instruction *range*() is a special function in **Python** that generates integer numbers included in a certain interval.
When it is combined with the *list*() function, it gives a list of integers.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

It is possible to give two or three arguments to the list function as shown:

```
>>> list(range(10,0,-1))
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

List of lists

This functionality called sometimes Matrix format is pratical in several ways.

List of lists

This functionality called sometimes Matrix format is pratical in several ways.

```
In [26]: L1=['lion',4]
         L2=['tigre',3]
         L3=['zebra',5]
         Z=[L1,L2,L3]

In [27]: Z

Out[27]: [['lion', 4], ['tigre', 3], ['zebra', 5]]

In [28]: Z[0]

Out[28]: ['lion', 4]

In [29]: Z[1]

Out[29]: ['tigre', 3]

In [30]: Z[2]

Out[30]: ['zebra', 5]
```

List of lists

This functionality called sometimes Matrix format is pratical in several ways.

```
In [26]: L1=['lion',4]
         L2=['tigre',3]
         L3=['zebra',5]
         Z=[L1,L2,L3]

In [27]: Z

Out[27]: [['lion', 4], ['tigre', 3], ['zebra', 5]]

In [28]: Z[0]

Out[28]: ['lion', 4]

In [29]: Z[1]

Out[29]: ['tigre', 3]

In [30]: Z[2]

Out[30]: ['zebra', 5]
```

In order to reach an element of the sub-list, we need a double indexing.

List of lists

This functionality called sometimes Matrix format is pratical in several ways.

```
In [26]: L1=['lion',4]
         L2=['tigre',3]
         L3=['zebra',5]
         Z=[L1,L2,L3]

In [27]: Z

Out[27]: [['lion', 4], ['tigre', 3], ['zebra', 5]]

In [28]: Z[0]

Out[28]: ['lion', 4]

In [29]: Z[1]

Out[29]: ['tigre', 3]

In [30]: Z[2]

Out[30]: ['zebra', 5]
```

In order to reach an element of the sub-list, we need a double indexing.

```
In [31]: Z[2][0]

Out[31]: 'zebra'

In [32]: Z[1][1]

Out[32]: 3
```

Exercise

Create 4 lists called Autumn, Winter, Spring and Summer that contain each all the corresponding months. Create a list S that contains the four previous lists. Predict then verify the output of each expression:

$$S[2],\ S[1][0],\ S[1:2] \textit{ and } S[:][1].$$

```
In [33]: Autumn=['september','october','november']
         Winter=['december','january','february']
         Spring=['march','april','may']
         Summer=['june','july','august']
         S=[Autumn,Winter,Spring,Summer]

In [34]: S

Out[34]: [['september', 'october', 'november'],
          ['december', 'january', 'february'],
          ['march', 'april', 'may'],
          ['june', 'july', 'august']]

In [35]: S[2]

Out[35]: ['march', 'april', 'may']

In [36]: S[1][0]

Out[36]: 'december'

In [37]: S[1:2]

Out[37]: [['december', 'january', 'february']]

In [38]: S[:][1]

Out[38]: ['december', 'january', 'february']
```

2.Loops and comparision

Loop FOR

In programming, we mostly need to repeat an instruction several times. That's why we need LOOPS.

Loop FOR

In programming, we mostly need to repeat an instruction several times. That's why we need LOOPS.

```
In [40]: M=['lion','tigre', 'monkey', 'zebra', 'bear', 'snake']

In [43]: print(M[0])

lion

In [44]: print(M[1])

tigre

In [45]: print(M[2])

monkey

In [46]: print(M[3])

zebra

In [47]: for i in M:
             print(i)

lion
tigre
monkey
zebra
bear
snake
```

- The letter i in the loop for is called ITERATION variable that changes its value at each iteration of the loop.

- The letter *i* in the loop for is called ITERATION variable that changes its value at each iteration of the loop.
- It is a dummy variable , which means can be replaced by anything else.

```
In [48]: for animal in M:
             print(animal)

lion
tigre
monkey
zebra
bear
snake
```

- The letter *i* in the loop for is called ITERATION variable that changes its value at each iteration of the loop.
- It is a dummy variable , which means can be replaced by anything else.
- The iteration variable can be of any type depending on the list.

- The letter *i* in the loop for is called ITERATION variable that changes its value at each iteration of the loop.
- It is a dummy variable , which means can be replaced by anything else.
- The iteration variable can be of any type depending on the list.
- We caracterize it by the <:> at the end of the line. This means that the loop for is waiting for a BLOC of instructions.

- The letter *i* in the loop for is called ITERATION variable that changes its value at each iteration of the loop.
- It is a dummy variable , which means can be replaced by anything else.
- The iteration variable can be of any type depending on the list.
- We caracterize it by the <:> at the end of the line. This means that the loop for is waiting for a BLOC of instructions.
- This bloc is considered as the body of the loop.

- The letter *i* in the loop for is called ITERATION variable that changes its value at each iteration of the loop.
- It is a dummy variable , which means can be replaced by anything else.
- The iteration variable can be of any type depending on the list.
- We caracterize it by the <:> at the end of the line. This means that the loop for is waiting for a BLOC of instructions.
- This bloc is considered as the body of the loop.
- In order to know where the bloc starts or ends , we use the identation which is made of the 4 spaces (1 TAB) with respect to the position of the word FOR.,

Example

```
In [49]: for animal in M :
             print ( animal )
             print ( animal *2)
         print (" it is over ")
```

Example

```
In [49]: for animal in M :
             print ( animal )
             print ( animal *2)
         print (" it is over ")
```

```
lion
lionlion
tigre
tigretigre
monkey
monkeymonkey
zebra
zebrazebra
bear
bearbear
snake
snakesnake
 it is over
```

The iteration can be over different types of lists and sub-lists:

The iteration can be over different types of lists and sub-lists:

```
In [50]: for animal in M[1:3]:
             print(animal)

tigre
monkey
```

The iteration can be over different types of lists and sub-lists:

```
In [50]: for animal in M[1:3]:
             print(animal)

tigre
monkey
```

```
In [51]: for i in [1,2,3]:
             print(i)

1
2
3
```

The iteration can be over different types of lists and sub-lists:

In [50]:
```
for animal in M[1:3]:
    print(animal)
```
```
tigre
monkey
```

In [51]:
```
for i in [1,2,3]:
    print(i)
```
```
1
2
3
```

In [52]:
```
for i in range(4):
    print(i)
```
```
0
1
2
3
```

The iteration can be over different types of lists and sub-lists:

```
In [50]: for animal in M[1:3]:
             print(animal)

tigre
monkey
```

```
In [51]: for i in [1,2,3]:
             print(i)

1
2
3
```

```
In [52]: for i in range(4):
             print(i)

0
1
2
3
```

```
In [53]: for i in range(3):
             print(M[i])

lion
tigre
monkey
```

The iteration can be over different types of lists and sub-lists:

```
In [50]: for animal in M[1:3]:
             print(animal)

tigre
monkey
```

```
In [51]: for i in [1,2,3]:
             print(i)

1
2
3
```

```
In [52]: for i in range(4):
             print(i)

0
1
2
3
```

```
In [53]: for i in range(3):
             print(M[i])

lion
tigre
monkey
```

```
In [56]: for i in range ( len ( M )):
             print (" The animal {} is a {}". format (i , M [ i ]))

The animal 0 is a lion
The animal 1 is a tigre
The animal 2 is a monkey
The animal 3 is a zebra
The animal 4 is a bear
The animal 5 is a snake
```

Guess!!

```
>>> for x in range(5):
        print(x)
```

Guess!!

```
>>> for x in range(5):
        print(x)



0
1
2
3
4
```

Guess!!

```
>>> for x in range(2,19,3):
        print(x)
```

Guess!!

```
>>> for x in range(2,19,3):
...     print(x)
...
2
5
8
11
14
17
```

Guess!!

```
>>> for x in range(2,19,3):
        print(2*x)
```

Guess!!

```
>>> for x in range(2,19,3):
...     print(2*x)
...
4
10
16
22
28
34
```

Example

```
In [40]: M=['lion','tigre', 'monkey', 'zebra', 'bear', 'snake']
```

```
In [49]: for animal in M :
             print ( animal )
             print ( animal *2)
         print (" it is over ")
```

Example

```
In [40]: M=['lion','tigre', 'monkey', 'zebra', 'bear', 'snake']
```

```
In [49]: for animal in M :
             print ( animal )
             print ( animal *2)
         print (" it is over ")
```

```
lion
lionlion
tigre
tigretigre
monkey
monkeymonkey
zebra
zebrazebra
bear
bearbear
snake
snakesnake
 it is over
```

The iteration can be over different types of lists and sub-lists:

The iteration can be over different types of lists and sub-lists:

```
In [50]: for animal in M[1:3]:
             print(animal)

tigre
monkey
```

The iteration can be over different types of lists and sub-lists:

In [50]:
```
for animal in M[1:3]:
    print(animal)
```

```
tigre
monkey
```

In [51]:
```
for i in [1,2,3]:
    print(i)
```

```
1
2
3
```

The iteration can be over different types of lists and sub-lists:

In [50]:
```
for animal in M[1:3]:
    print(animal)
```
```
tigre
monkey
```

In [51]:
```
for i in [1,2,3]:
    print(i)
```
```
1
2
3
```

In [52]:
```
for i in range(4):
    print(i)
```
```
0
1
2
3
```

The iteration can be over different types of lists and sub-lists:

```
In [50]: for animal in M[1:3]:
             print(animal)

tigre
monkey
```

```
In [51]: for i in [1,2,3]:
             print(i)

1
2
3
```

```
In [52]: for i in range(4):
             print(i)

0
1
2
3
```

```
In [53]: for i in range(3):
             print(M[i])

lion
tigre
monkey
```

The iteration can be over different types of lists and sub-lists:

```
In [50]: for animal in M[1:3]:
             print(animal)

tigre
monkey
```

```
In [51]: for i in [1,2,3]:
             print(i)

1
2
3
```

```
In [52]: for i in range(4):
             print(i)

0
1
2
3
```

```
In [53]: for i in range(3):
             print(M[i])

lion
tigre
monkey
```

```
In [56]: for i in range ( len ( M )):
             print (" The animal {} is a {}". format (i , M [ i ]))

The animal 0 is a lion
The animal 1 is a tigre
The animal 2 is a monkey
The animal 3 is a zebra
The animal 4 is a bear
The animal 5 is a snake
```

Guess!!

Explain the following code and guess the result:

Guess!!

Explain the following code and guess the result:

```
>>> result=[]
>>> for i in range(6):
...     result.append(i+3)
```

Guess!!

Explain the following code and guess the result:

```
>>> result=[]
>>> for i in range(6):
...     result.append(i+3)
```

```
>>> result
[3, 4, 5, 6, 7, 8]
```

Guess!!

Explain the following code and guess the result:

```
>>> result=[]
>>> for i in range(6):
...     result.append(i+3)
```

```
>>> result
[3, 4, 5, 6, 7, 8]
```

Write it in a function

Guess!!

Explain the following code and guess the result:

```
>>> result=[]
>>> for i in range(6):
...     result.append(i+3)
```

```
>>> result
[3, 4, 5, 6, 7, 8]
```

Write it in a function

```
>>> def guess(n):
...     l=[]
...     for i in range(n):
...         l.append(i)
...     return(l)
```

Guess!!

Explain the following code and guess the result:

```
>>> result=[]
>>> for i in range(6):
...     result.append(i+3)
```

```
>>> result
[3, 4, 5, 6, 7, 8]
```

Write it in a function

```
>>> def guess(n):
...     l=[]
...     for i in range(n):
...         l.append(i)
...     return(l)
```

```
...
>>> guess(5)
[0, 1, 2, 3, 4]
>>> guess(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> guess(15)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>>
```

Guess!!

Explain the following code and guess the result:

```
>>> result=[]
>>> for i in range(6):
...     result.append(i+3)
```

```
>>> result
[3, 4, 5, 6, 7, 8]
```

Write it in a function

```
>>> def guess(n):
...     l=[]
...     for i in range(n):
...         l.append(i)
...     return(l)
```

```
...
>>> guess(5)
[0, 1, 2, 3, 4]
>>> guess(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> guess(15)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>>
```

What does each of the codes do? What is the diffrence between them?

Guess!!

Explain the following code and guess the result:

```
>>> result=[]
>>> for i in range(6):
...     result.append(i+3)
```

```
>>> result
[3, 4, 5, 6, 7, 8]
```

Write it in a function

```
>>> def guess(n):
...     l=[]
...     for i in range(n):
...         l.append(i)
...     return(l)
```

```
>>> guess(5)
[0, 1, 2, 3, 4]
>>> guess(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> guess(15)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>>
```

What does each of the codes do? What is the diffrence between them?

```
for i in range(6):
    if i >= 2:
        result3.append(i+3)
```

```
result4 = [i+3 for i in range(6) if i >= 2]
```

Exercise 1

Write a function that returns the following sum:

$$\Sigma_1^n i = 1 + 2 + 3 \ldots + n.$$

Exercise 1

Write a function that returns the following sum:

$$\Sigma_1^n i = 1 + 2 + 3 \ldots + n.$$

```
In [6]: def summation(n):
            a=0
            for i in range(n):
                a+=i+1
            return a

In [7]: summation(6)

Out[7]: 21

In [3]: 1+2+3+4+5+6

Out[3]: 21
```

Exercise 2

Try to write the factorial function this time using the Loop FOR:

Exercise 2

Try to write the factorial function this time using the Loop FOR:

```
In [11]: def factorial(n):
             k=1
             for i in range(n):
                 k*=(i+1)
             return(k)

In [12]: factorial(5)

Out[12]: 120

In [13]: factorial(7)

Out[13]: 5040

In [14]: factorial(3)

Out[14]: 6
```

Exercise 3

Write a function that tests the parity of a given number *n*.

Exercise 3

Write a function that tests the parity of a given number *n*.

```
In [15]: def parity(n):
             if n%2==0:
                 return('The number is even')
             else:
                 return('The number is odd')

In [16]: parity(5)

Out[16]: 'The number is odd'

In [17]: parity(102)

Out[17]: 'The number is even'
```

Exercise 4

Create a function that transforms the integer elements of a list into there squares.

Exercise 4

Create a function that transforms the integer elements of a list into there squares.

```
In [1]: def square(l):
            p=[]
            n=len(l)
            for i in range(n):
                p.append((l[i])**2)
            return p

In [2]: l=[1,2,3,4,5,6,7,8,9,10]

In [3]: square(l)

Out[3]: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Exercise 5

Create a function that displays a list of numerical values of how fast e^{-x} converges to 0.

Exercise 5

Create a function that displays a list of numerical values of how fast e^{-x} converges to 0.

```
In [4]: import math
        def f(n):
            l=[]
            for i in range(n+1):
                l.append(math.e**(-i))
            return l

In [5]: f(5)

Out[5]: [1.0,
         0.36787944117144233,
         0.1353352832366127,
         0.049787068367863944,
         0.018315638888734186,
         0.006737946999085469]

In [6]: f(10)

Out[6]: [1.0,
         0.36787944117144233,
         0.1353352832366127,
         0.049787068367863944,
         0.018315638888734186,
         0.006737946999085469,
         0.0024787521766663594,
         0.0009118819655545166,
         0.00033546262790251196,
         0.00012340980408667962,
         4.5399929762484875e-05]
```

Loop WHILE

While-loops are **loops** that repeat while/until a specific condition is met.

Loop WHILE

While-loops are **loops** that repeat while/until a specific condition is met. They can replace for-loops for quicker computation in a lot of cases.

Loop WHILE

While-loops are **loops** that repeat while/until a specific condition is met.
They can replace for-loops for quicker computation in a lot of cases.
Additionally, they do not require a **parameter** for the amount of repetitions.

Loop WHILE

While-loops are **loops** that repeat while/until a specific condition is met.
They can replace for-loops for quicker computation in a lot of cases.
Additionally, they do not require a **parameter** for the amount of repetitions.
While-loops simply check the condition statement at each repetition of the loop.

Loop WHILE

While-loops are **loops** that repeat while/until a specific condition is met.
They can replace for-loops for quicker computation in a lot of cases.
Additionally, they do not require a **parameter** for the amount of repetitions.
While-loops simply check the condition statement at each repetition of the loop.
If the conditional statement is not met, the loop breaks.

Loop WHILE

While-loops are **loops** that repeat while/until a specific condition is met.
They can replace for-loops for quicker computation in a lot of cases.
Additionally, they do not require a **parameter** for the amount of repetitions.
While-loops simply check the condition statement at each repetition of the loop.
If the conditional statement is not met, the loop breaks.
The syntax is: while condition :

```
>>> l = []
>>> x = 0
>>> while x < 100:
        l.append(x)
        x += 1
>>> print(l)

[1,2,3,4,5, ...  51,52,53, ...  97, 98, 99]
```

```
>>> l = []
>>> x = 0
>>> while x < 100:
        l.append(x)
        x += 1
>>> print(l)
[1,2,3,4,5, ...  51,52,53, ...  97, 98, 99]
```

Note

If the fifth line of $x+=1$ hadn't been added, the loop would continue for ever, as the loop would check $0 < 100$ and then afterwards the loop would append(x) to l.

```
>>> l = []
>>> x = 0
>>> while x < 100:
        l.append(x)
        x += 1
>>> print(l)
[1,2,3,4,5, ...  51,52,53, ...  97, 98, 99]
```

Note

If the fifth line of $x+=1$ hadn't been added, the loop would continue for ever, as the loop would check $0 < 100$ and then afterwards the loop would append(x) to l.

Therefore, it is mandatory to think ahead and ensure that the condition that is being checked is **continuously changing**, otherwise, the loop will never break.

BREAK

1. It is a command that can be used in for-loops to imitate a while-loop.

BREAK

1. It is a command that can be used in for-loops to imitate a while-loop.
2. Break can also be used in a lot of other contexts, as what it does is as soon as a condition is met, it will stop the loop or the function.

BREAK

1. It is a command that can be used in for-loops to imitate a while-loop.
2. Break can also be used in a lot of other contexts, as what it does is as soon as a condition is met, it will stop the loop or the function.

```
>>> k = []
>>> for x in range(1000):
        if x>= 100:
                break
        else:
                k.append(x)
>>>print(k)
[1,2,3,4,5, ...  51,52,53, ...  97, 98, 99]
```

BREAK

1. It is a command that can be used in for-loops to imitate a while-loop.
2. Break can also be used in a lot of other contexts, as what it does is as soon as a condition is met, it will stop the loop or the function.

```
>>> k = []
>>> for x in range(1000):
        if x>= 100:
                break
        else:
                k.append(x)
>>>print(k)
[1,2,3,4,5, ...  51,52,53, ...  97, 98, 99]
```

Note

It is seen that both of the two scripts before yield the natural numbers up to, but not including 100. The reason for this is the com- mand **break**, and hopefully this illustration will help better understand while-loops. While loops are computationally less requiring, and the syntax is easier, thus the for-loop with break should be avoided.

Exercise 6

Create a script that displays all of the values of $2^n < 106$, where $n \in N$.

Exercise 6

Create a script that displays all of the values of $2^n < 106$, where $n \in N$.

```
def f(x):
        return 2**x
bound = 10**6
n = 0
l = []
while f(n) < bound:
        l.append(f(n))
        n += 1

print(l)
```

```
>>>
```
$[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096,$
$8192, 16384, 32768, 65536, 131072, 262144, 524288]$

Exercise 7

Write a function that deduces the factorial of any number $n \in N$, only using a while loop.

Exercise 7

Write a function that deduces the factorial of any number $n \in N$, only using a while loop.

```
In [7]: def fac3(n):
            k=1
            while n>=1:
                k=k*n
                n-=1
            return k

In [8]: fac3(5)

Out[8]: 120

In [9]: fac3(7)

Out[9]: 5040
```

Exercise 8

Write a function using a while loop that gives the n^{th} element of the following numerical sequence, where $u_0 = 7$ and $u_n = 2u_{n-1} + 3$ *for* $n > 0$.

Exercise 8

Write a function using a while loop that gives the n^{th} element of the following numerical sequence, where $u_0 = 7$ and $u_n = 2u_{n-1} + 3$ *for* $n > 0$.

```
In [11]: def seq_u(n):
             u=7
             i=0
             while i<n:
                 u=2*u+3
                 i=i+1
             return u

In [12]: seq_u(0)

Out[12]: 7

In [13]: seq_u(1)

Out[13]: 17

In [14]: seq_u(5)

Out[14]: 317
```

Exercise 9

Write a function using a while loop that gives the sum of the following elemnts:

$$S_n = \Sigma_{k\geq 0}^{n}(\frac{1}{2})^k.$$

Exercise 9

Write a function using a while loop that gives the sum of the following elemnts:

$$S_n = \Sigma_{k\geq 0}^{n}(\frac{1}{2})^k.$$

```
In [15]: def serie(n):
             s=0.0
             k=0
             while k<=n:
                 s=s+((1/2)**k)
                 k=k+1
             return s

In [16]: serie(3)

Out[16]: 1.875

In [17]: serie(5)

Out[17]: 1.96875

In [18]: serie(1)

Out[18]: 1.5

In [19]: serie(1000)

Out[19]: 2.0
```

Exercise 10

Write a function using a while loop that gives the GCD between two positive integers.

Exercise 10

Write a function using a while loop that gives the GCD between two positive integers.

Hint

Using Euclide algorithm, we know that to calculate $gcd(a, b)$; $a \geq b$:

- *if* $b \neq 0$ *then* $GCD(a, b) = GCD(b, a\%b)$
- *Else, the* $GCD(a, b) = a$
  Remark: we can easily verify that if $a \geq b; b \neq 0$, *then* $b \geq a\%b$.

Exercise 10

Write a function using a while loop that gives the GCD between two positive integers.

Step n	**Dividend** r_{n-1}	**Diviseur** r_n	**Equation** $r_{n-1} = r_n q_n + r_{n+1}$	**Quotient** q_n	**Rest** r_{n+1}
1	21	15	21 = 15 × 1 + 6	1	6
2	15	6	15 = 6 × 2 + 3	2	3
3	6	3	6 = 3 × 2 + 0	2	0
4	3	0	End of the algorithm		

Exercise 10

Write a function using a while loop that gives the GCD between two positive integers.

$gcd(4052, 420) = 4$

$$4052 = 9 \times 420 + 272$$

$$420 = 1 \times 272 + 148$$

$$272 = 1 \times 148 + 124$$

$$148 = 1 \times 124 + 24$$

$$124 = 5 \times 24 + 4$$

$$24 = 6 \times 4 + 0$$

Solution

In [20]:
```
def pgcd(a,b):
    q=a #the quotient
    r=b #divisor
    t=0
    while r!=0:
        t=q%r
        q=r
        r=t
    return q
```

In [21]:
```
pgcd(9,3)
```
Out[21]: 3

In [24]:
```
pgcd(15,15)
```
Out[24]: 15

In [23]:
```
pgcd(56,42)
```
Out[23]: 14

In [25]:
```
pgcd(4199, 1530)
```
Out[25]: 17

Exercise 11

Create a scriot that can show the following Triangle:

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Exercise 11

Create a scriot that can show the following Triangle:

```
In [3]: star='*'
        for i in range(10):
            print((i+1)*star)
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Exercise 12

Create a scriot that can show the following Triangle:

```
**********
*********
********
*******
******
*****
****
***
**
*
```

Exercise 12

Create a scriot that can show the following Triangle:

In [3]:
```
star='*'
i=10
while i>=1:
    print(i*star)
    i-=1
```

```
**********
*********
********
*******
******
*****
****
***
**
*
```

Exercise 13

Create a scriot that can show the following Triangle:

```
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

Exercise 13

Create a scriot that can show the following Triangle:

```
In [4]: space=' '
        star='*'
        for i in range(10):
            print((9-i)*space+(i+1)*star)
```

```
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

Exercise 14

Create a function that can show the following Pyramid for any number of lines:

```
         *
        ***
       *****
      *******
     *********
    ***********
   *************
  ***************
 *****************
*******************
```

Exercise 14

Create a function that can show the following Pyramid for any number of lines:

```
In [11]: def pyra(n):
             space=' '
             star='*'
             for i in range(1,10):
                 print((n-i)*space+(2*i-1)*star)
```

```
         *
        ***
       *****
      *******
     *********
    ***********
   *************
  ***************
 *****************
*******************
```

2.Matplot library

Matplotlib.pyplot

- Matplotlib is an enormous module that has functions for plotting data in $2D$ and $3D$.

Matplotlib.pyplot

- Matplotlib is an enormous module that has functions for plotting data in $2D$ and $3D$.
- Some of the functions include plot(input, output), ylabel(''), title('').

Matplotlib.pyplot

- Matplotlib is an enormous module that has functions for plotting data in $2D$ and $3D$.
- Some of the functions include plot(input, output), ylabel(''), title('').
- The syntax and arguments in matplotlib can seem daunting, however, they are in fact very simple with a slight amount of practice.

Example

1. import matplotlib.pyplot as plt
   plt.plot([1, 2, 3, 4])
   plt.ylabel('some numbers')
   plt.show()

Example

1. ```
   import matplotlib.pyplot as plt
   plt.plot([1, 2, 3, 4])
   plt.ylabel('some numbers')
   plt.show()
   ```

2. ```
   plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
   ```

Example

1. ```
   import matplotlib.pyplot as plt
   plt.plot([1, 2, 3, 4])
   plt.ylabel('some numbers')
   plt.show()
   ```
2. ```
   plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
   ```
3. ```
   plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
   plt.axis([0, 6, 0, 20])
   plt.show()
   ```
   ```

Example

1. import matplotlib.pyplot as plt
   plt.plot([1, 2, 3, 4])
   plt.ylabel('some numbers')
   plt.show()

2. plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
3. plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
   plt.axis([0, 6, 0, 20])
   plt.show()

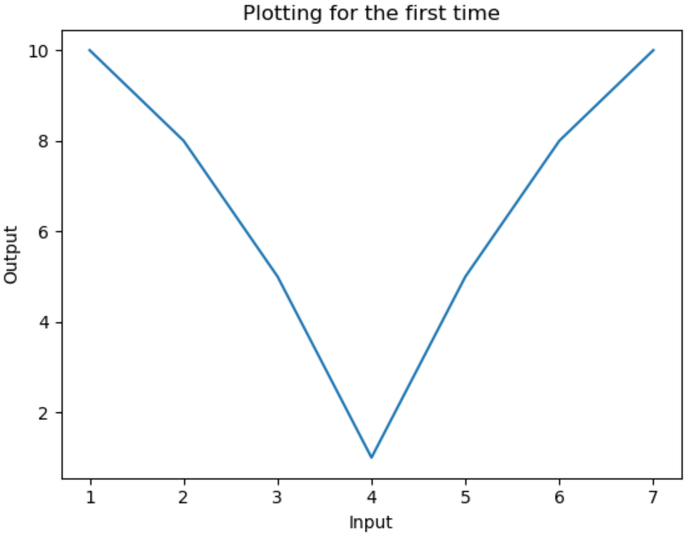
```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

4.

```
In [2]: import matplotlib.pyplot as plt
   ...: x=[1,2,3,4,5,6,7]
   ...: y=[10,8,5,1,5,8,10]
   ...: plt.title('Plotting for the first time')
   ...: plt.ylabel('Output')
   ...: plt.xlabel('Input')
   ...: plt.plot(x,y)
   ...:
Out[2]: [<matplotlib.lines.Line2D at 0x7f96086233d0>]
```


Plotting for the first time
10
8
6
4
2
Output
1
2
3
4
5
6
7
Input

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

1. 'r' makes the color red.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

1. 'r' makes the color red.
2. 'b' makes the color blue .

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

1. 'r' makes the color red.
2. 'b' makes the color blue .
3. 'k' makes the graph **black**.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

1. 'r' makes the color red.
2. 'b' makes the color blue .
3. 'k' makes the graph **black**.
4. ':' makes the line dotted.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

1. 'r' makes the color red.
2. 'b' makes the color blue .
3. 'k' makes the graph **black**.
4. ':' makes the line dotted.
5. 'o' makes the graph a scatter plot with circles.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

1. 'r' makes the color red.
2. 'b' makes the color blue .
3. 'k' makes the graph **black**.
4. ':' makes the line dotted.
5. 'o' makes the graph a scatter plot with circles.
6. '+' makes the graph a scatter plot with '+' as points.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

1. 'r' makes the color red.
2. 'b' makes the color blue .
3. 'k' makes the graph **black**.
4. ':' makes the line dotted.
5. 'o' makes the graph a scatter plot with circles.
6. '+' makes the graph a scatter plot with '+' as points.
7. 'k:' makes the graph a black dotted line.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

1. 'r' makes the color red.
2. 'b' makes the color blue .
3. 'k' makes the graph **black**.
4. ':' makes the line dotted.
5. 'o' makes the graph a scatter plot with circles.
6. '+' makes the graph a scatter plot with '+' as points.
7. 'k:' makes the graph a black dotted line.
8. '.r' makes the graph a scatter plot with small red points.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

1. 'r' makes the color red.
2. 'b' makes the color blue .
3. 'k' makes the graph **black**.
4. ':' makes the line dotted.
5. 'o' makes the graph a scatter plot with circles.
6. '+' makes the graph a scatter plot with '+' as points.
7. 'k:' makes the graph a black dotted line.
8. '.r' makes the graph a scatter plot with small red points.

Note that:This list is not a full list of all of the possible commands.

More details

The full syntax for this module command is as follows:

$$plt.plot(input, output, type, label = label).$$

The syntax is **trivial**, except for 'type'.
The list below shows the possible inputs and the function of the input.

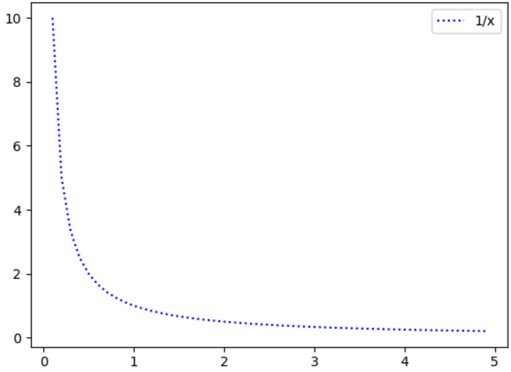
1. 'r' makes the color red.
2. 'b' makes the color blue .
3. 'k' makes the graph **black**.
4. ':' makes the line dotted.
5. 'o' makes the graph a scatter plot with circles.
6. '+' makes the graph a scatter plot with '+' as points.
7. 'k:' makes the graph a black dotted line.
8. '.r' makes the graph a scatter plot with small red points.

Note that:This list is not a full list of all of the possible commands.
The arguments inside the string can be concatenated to yield a graph with all of the parameters requested as in 6 and 7.

In [7]:

```python
import matplotlib.pyplot as plt
import numpy as np
def f(x):
    return 1/x
x= np.arange(0.1, 5, 0.1)
plt.plot(x, f(x), 'b:',label = '1/x $')
plt.legend()
```

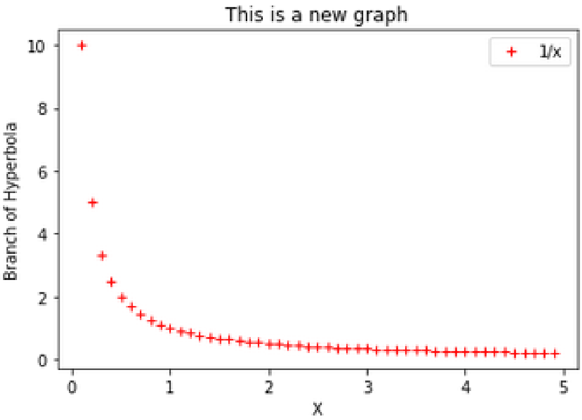
In [7]:
```python
import matplotlib.pyplot as plt
import numpy as np
def f(x):
    return 1/x
x= np.arange(0.1, 5, 0.1)
plt.plot(x, f(x), 'b:',label = '1/x $')
plt.legend()
```



Some modifications

In [15]:
```
import matplotlib.pyplot as plt
import numpy as np
def f(x):
    return 1/x
x= np.arange(0.1, 5, 0.1)
plt.title('This is a new graph')
plt.ylabel('Branch of Hyperbola')
plt.xlabel('X')
plt.plot(x, f(x), 'r+',label = '1/x')
plt.legend()
```

Some modifications



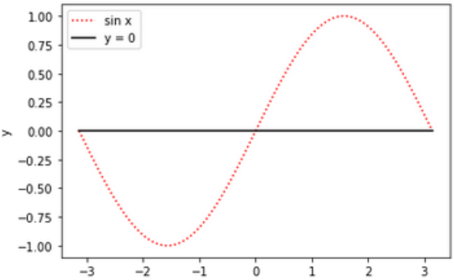
Exercise I

Write a script that will produce a red curve of *sinx* in the interval of $[-\pi, \pi]$. Ensure that there is a black $x - axis$.

Exercise I

Write a script that will produce a red curve of *sinx* in the interval of $[-\pi, \pi]$. Ensure that there is a black $x - axis$.

```
In [16]: import numpy as np
         import matplotlib.pyplot as plt
         def f(x):
             return np.sin(x)
         x = np.arange(- np.pi, np.pi, 0.01)
         plt.ylabel("y")
         plt.xlabel("x")
         plt.plot(x, f(x), "r:", label = "sin x")
         plt.plot(x, x*0, "k", label = "y = 0")
         plt.legend()

Out[16]: <matplotlib.legend.Legend at 0x7f8651a862e0>
```



Exercise II

Make a script that approximates the limit of f as $x \to 0$ of $\sin(x)$ defined as:

$$f :]0, 1] \to \mathbb{R} : x \mapsto \frac{\sin x}{x}$$

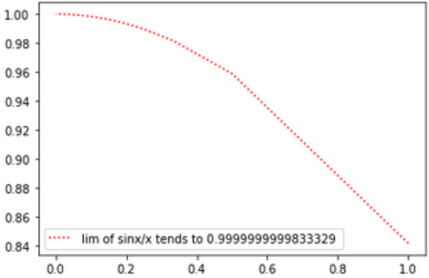
Exercise II

Make a script that approximates the limit of f as $x \to 0$ of $\sin(x)$ defined as:

$$f :]0,1] \to \mathbb{R} : x \mapsto \frac{\sin x}{x}$$

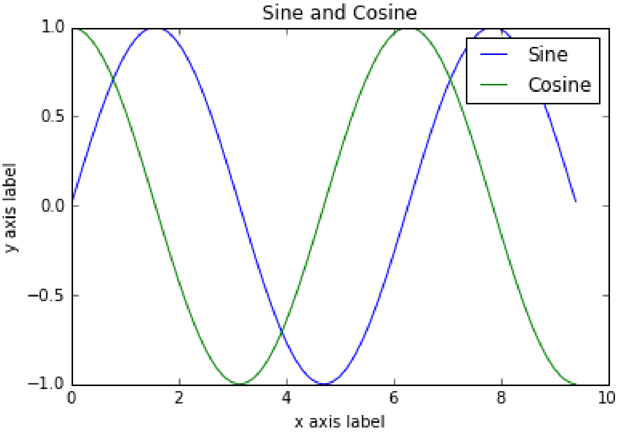
In [32]:
```
import numpy as np
import matplotlib.pyplot as plt
N =100000
i=np.arange(1,N)
x=1/i
y=np.sin(x)/x
plt.plot(x,y, 'r:', label = "lim of sinx/x tends to 0.9999999999833329 ")
plt.legend()
```

Out[32]: <matplotlib.legend.Legend at 0x7f86519491f0>



Exercise III

Write the code that gives you the following figure.



Exercise III

Write the code that gives you the following figure.

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```

Interlude

Create an array of equally spaced 10 data in range 0 to 2π.

Interlude

Create an array of equally spaced 10 data in range 0 to 2π.

In [4]:
```
import numpy as np
l=[]
for i in range(10):
    l.append(i/10*np.pi*2)
print(l)
```

```
[0.0, 0.6283185307179586, 1.2566370614359172, 1.8849555921538759, 2.5132741228718345, 3.141592653589793, 3.76991118
43077517, 4.39822971502571, 5.026548245743669, 5.654866776461628]
```

Interlude

Create an array of equally spaced 10 data in range 0 to 2π.

```
In [4]: import numpy as np
        l=[]
        for i in range(10):
            l.append(i/10*np.pi*2)
        print(l)

[0.0, 0.6283185307179586, 1.2566370614359172, 1.8849555921538759, 2.5132741228718345, 3.141592653589793, 3.76991118
43077517, 4.39822971502571, 5.026548245743669, 5.654866776461628]
```

Another solution is using numpy linspace

```
In [7]: print(np.linspace(0,2*np.pi,11))

[0.         0.62831853 1.25663706 1.88495559 2.51327412 3.14159265
 3.76991118 4.39822972 5.02654825 5.65486678 6.28318531]
```

Exercise IV

This example uses following formula to generate heart shape x and y coordinates:

$$x = 16 * sin^3(\theta) \ and \ y = 13 * cos(\theta) - 5 * cos(2\theta) - 2 * cos(3\theta) - cos(4\theta)$$

Exercise IV

This example uses following formula to generate heart shape x and y coordinates:

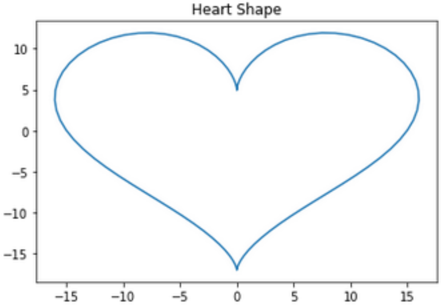
$$x = 16 * sin^3(\theta) \text{ and } y = 13 * cos(\theta) - 5 * cos(2\theta) - 2 * cos(3\theta) - cos(4\theta)$$

```
# Python program to Plot Perfect Heart Shape

# importing libraries
import numpy as np
from matplotlib import pyplot as plt

# Creating equally spaced 100 data in range 0 to 2*pi
theta = np.linspace(0, 2 * np.pi, 100)

# Generating x and y data
x = 16 * ( np.sin(theta) ** 3 )
y = 13 * np.cos(theta) - 5* np.cos(2*theta) - 2 * np.cos(3*theta) - np.cos(4*theta)

# Plotting
plt.plot(x, y)
plt.title('Heart Shape')
plt.show()
```

Exercise IV

This example uses following formula to generate heart shape x and y coordinates:

$$x = 16 * sin^3(\theta) \text{ and } y = 13 * cos(\theta) - 5 * cos(2\theta) - 2 * cos(3\theta) - cos(4\theta)$$



This part is for the documentation. Check this section in particular:
matplotlib.pyplot.plot(*args, **kwargs)

Subplots

You can plot different things in the same figure using the subplot function. Here is an example:

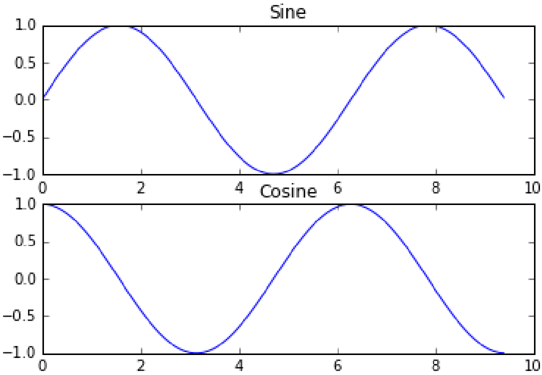
Subplots

You can plot different things in the same figure using the subplot function. Here is an example:

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```

Subplots

You can plot different things in the same figure using the subplot function. Here is an example:

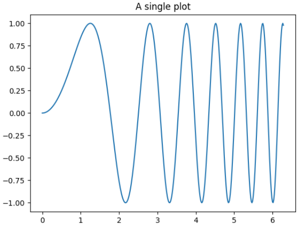


Second way with subplots

```
import matplotlib.pyplot as plt
import numpy as np

# Some example data to display
x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x ** 2)
```

subplots() without arguments returns a **Figure** and a **single Axes.**

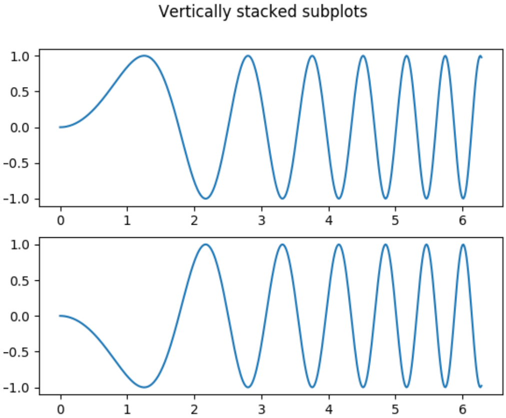
```
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_title('A single plot')
```



The first two optional arguments of pyplot.subplots define the number of rows and columns of the subplot grid.

The first two optional arguments of pyplot.subplots define the number of rows and columns of the subplot grid.

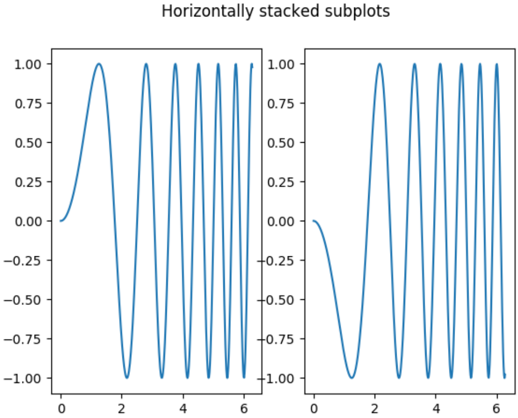
```
fig, axs = plt.subplots(2)
fig.suptitle('Vertically stacked subplots')
axs[0].plot(x, y)
axs[1].plot(x, -y)
```



To obtain side-by-side subplots, pass parameters $1, 2$ for one row and two columns.

To obtain side-by-side subplots, pass parameters $1, 2$ for one row and two columns.

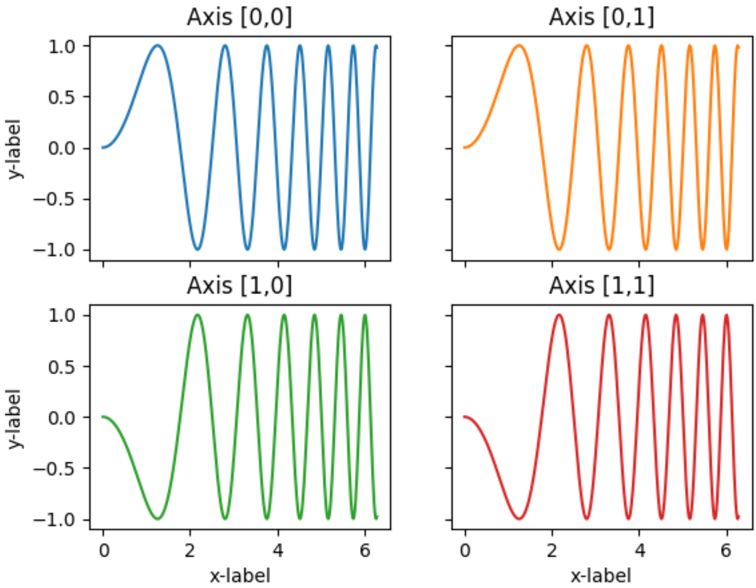
```
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Horizontally stacked subplots')
ax1.plot(x, y)
ax2.plot(x, -y)
```



Stacking subplots in two directions

```
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(x, y)
axs[0, 0].set_title('Axis [0,0]')
axs[0, 1].plot(x, y, 'tab:orange')
axs[0, 1].set_title('Axis [0,1]')
axs[1, 0].plot(x, -y, 'tab:green')
axs[1, 0].set_title('Axis [1,0]')
axs[1, 1].plot(x, -y, 'tab:red')
axs[1, 1].set_title('Axis [1,1]')

for ax in axs.flat:
    ax.set(xlabel='x-label', ylabel='y-label')

# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
```

Stacking subplots in two directions



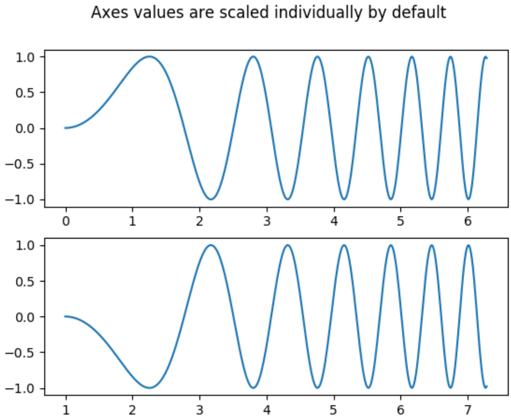
By default, each Axes is scaled individually. Thus, if the ranges are different the tick values of the subplots do not align.

By default, each Axes is scaled individually. Thus, if the ranges are different the tick values of the subplots do not align.

```
fig, (ax1, ax2) = plt.subplots(2)
fig.suptitle('Axes values are scaled individually by default')
ax1.plot(x, y)
ax2.plot(x + 1, -y)
```

By default, each Axes is scaled individually. Thus, if the ranges are different the tick values of the subplots do not align.

```
fig, (ax1, ax2) = plt.subplots(2)
fig.suptitle('Axes values are scaled individually by default')
ax1.plot(x, y)
ax2.plot(x + 1, -y)
```



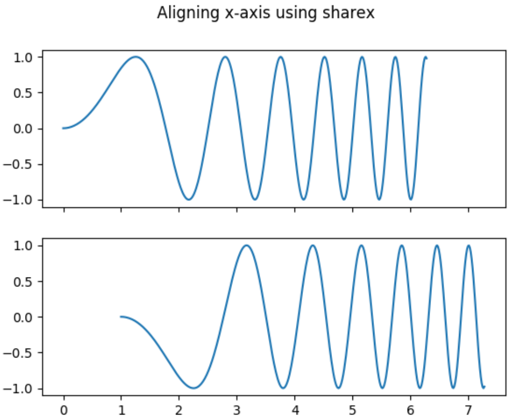
You can use sharex or sharey to align the **horizontal or vertical axis**.

You can use sharex or sharey to align the **horizontal or vertical axis**.

```
fig, (ax1, ax2) = plt.subplots(2, sharex=True)
fig.suptitle('Aligning x-axis using sharex')
ax1.plot(x, y)
ax2.plot(x + 1, -y)
```

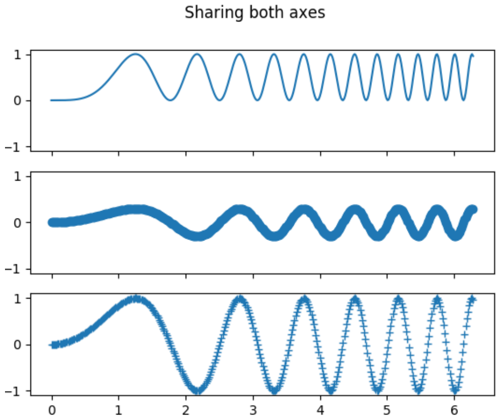
You can use sharex or sharey to align the **horizontal or vertical axis**.

```
fig, (ax1, ax2) = plt.subplots(2, sharex=True)
fig.suptitle('Aligning x-axis using sharex')
ax1.plot(x, y)
ax2.plot(x + 1, -y)
```



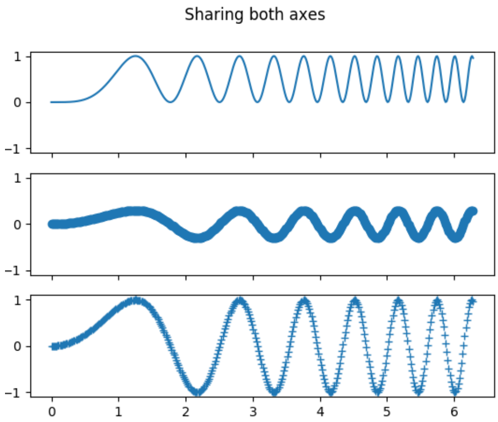
You can use sharex or sharey to align the **horizontal or vertical axis**.

```
fig, axs = plt.subplots(3, sharex=True, sharey=True)
fig.suptitle('Sharing both axes')
axs[0].plot(x, y ** 2)
axs[1].plot(x, 0.3 * y, 'o')
axs[2].plot(x, y, '+')
```



You can use sharex or sharey to align the **horizontal or vertical axis**.

```
fig, axs = plt.subplots(3, sharex=True, sharey=True)
fig.suptitle('Sharing both axes')
axs[0].plot(x, y ** 2)
axs[1].plot(x, 0.3 * y, 'o')
axs[2].plot(x, y, '+')
```

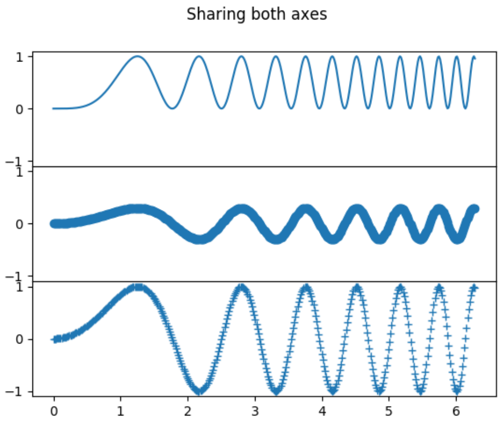


The parameter **gridspec_kw** of pyplot.subplots controls the grid properties. For example, we can reduce the height between vertical subplots using **gridspec_kw={'hspace': 0}**.
label_outer is a handy method to remove labels and ticks from subplots that are not at the edge of the grid.

The parameter **gridspec_kw** of pyplot.subplots controls the grid properties. For example, we can reduce the height between vertical subplots using **gridspec_kw={'hspace': 0}**.
label_outer is a handy method to remove labels and ticks from subplots that are not at the edge of the grid.

```
fig, axs = plt.subplots(3, sharex=True, sharey=True, gridspec_kw={'hspace': 0})
fig.suptitle('Sharing both axes')
axs[0].plot(x, y ** 2)
axs[1].plot(x, 0.3 * y, 'o')
axs[2].plot(x, y, '+')

# Hide x labels and tick labels for all but bottom plot.
for ax in axs:
    ax.label_outer()
```

The parameter **gridspec_kw** of pyplot.subplots controls the grid properties. For example, we can reduce the height between vertical subplots using **gridspec_kw={'hspace': 0}**.
label_outer is a handy method to remove labels and ticks from subplots that are not at the edge of the grid.

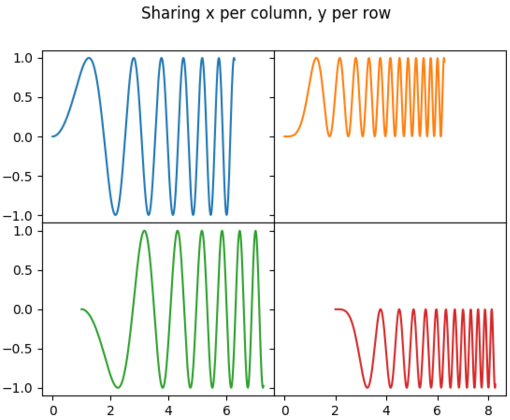


Apart from `True` and `False`, both `sharex` and `sharey` accept the values 'row' and 'col' to share the values only per row or column.

Apart from `True` and `False`, both `sharex` and `sharey` accept the values 'row' and 'col' to share the values only per row or column.

```
fig, axs = plt.subplots(2, 2, sharex='col', sharey='row',
                        gridspec_kw={'hspace': 0, 'wspace': 0})
(ax1, ax2), (ax3, ax4) = axs
fig.suptitle('Sharing x per column, y per row')
ax1.plot(x, y)
ax2.plot(x, y**2, 'tab:orange')
ax3.plot(x + 1, -y, 'tab:green')
ax4.plot(x + 2, -y**2, 'tab:red')

for ax in axs.flat:
    ax.label_outer()
```

Apart from `True` and `False`, both `sharex` and `sharey` accept the values 'row' and 'col' to share the values only per row or column.



3.3D plotting

3D plotting

Importing the `mplot3d` library enables 3D plotting.
You can create 3D axes by passing `projection="3d"` to any of the regular axes creation functions.

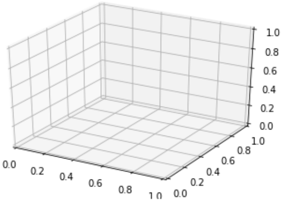
3D plotting

Importing the `mplot3d` library enables 3D plotting.
You can create 3D axes by passing `projection="3d"` to any of the regular axes creation functions.

```
from mpl_toolkits import mplot3d

import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.axes(projection="3d")

plt.show()
```

3D plotting

Importing the `mplot3d` library enables 3D plotting.
You can create 3D axes by passing `projection="3d"` to any of the regular axes creation functions.



Plotting a 3d curve

$$x = \cos(z) \tag{1}$$

$$y = \sin(z) \tag{2}$$

$$z = z \tag{3}$$

Plotting a 3d curve

$$x = \cos(z) \tag{1}$$

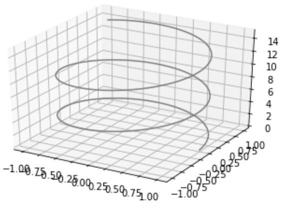
$$y = \sin(z) \tag{2}$$

$$z = z \tag{3}$$

```python
fig = plt.figure()
ax = plt.axes(projection="3d")

z_line = np.linspace(0, 15, 1000)
x_line = np.cos(z_line)
y_line = np.sin(z_line)
ax.plot3D(x_line, y_line, z_line, 'gray')

plt.show()
```

Plotting a 3d curve

$$x = \cos(z) \quad (1)$$
$$y = \sin(z) \quad (2)$$
$$z = z \quad (3)$$



Example: Plotting a 3d curve

Draw the 3D plot of

$$x = z \times \sin(20z) \tag{4}$$

$$y = z \times \cos(20z) \tag{5}$$

$$z = z \tag{6}$$

Example: Plotting a 3d curve

Draw the 3D plot of

$$x = z \times \sin(20z) \tag{4}$$
$$y = z \times \cos(20z) \tag{5}$$
$$z = z \tag{6}$$

```
fig = plt.figure()
ax = plt.axes(projection='3d')

z = np.linspace(0, 1, 100)
x = z * np.sin(20 * z)
y = z * np.cos(20 * z)

ax.plot3D(x, y, z, 'red')
```

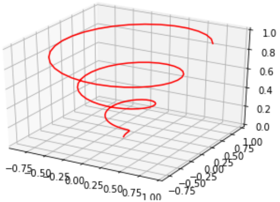
Example: Plotting a 3d curve

Draw the 3D plot of

$$x = z \times \sin(20z) \tag{4}$$

$$y = z \times \cos(20z) \tag{5}$$

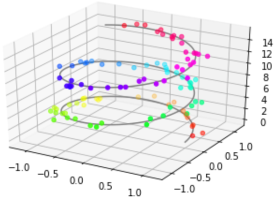
$$z = z \tag{6}$$



Plotting a 3d curve and some points

```
fig = plt.figure()
ax = plt.axes(projection="3d")

z_line = np.linspace(0, 15, 1000)
x_line = np.cos(z_line)
y_line = np.sin(z_line)
ax.plot3D(x_line, y_line, z_line, 'gray')

z_points = 15 * np.random.random(100)
x_points = np.cos(z_points) + 0.1 * np.random.randn(100)
y_points = np.sin(z_points) + 0.1 * np.random.randn(100)
ax.scatter3D(x_points, y_points, z_points, c=z_points, cmap='hsv');

plt.show()
```

Plotting a 3d curve and some points



1.Surfaces

Plotting a surface (1)

This is a 3 step process.
First step is to define the function and to generate enough points to estimate the surface.

```
def z_function(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)

X, Y = np.meshgrid(x, y)
Z = z_function(X, Y)
```

Plotting a surface (2)

The second step is to plot a wire-frame – this is our estimate of the surface.

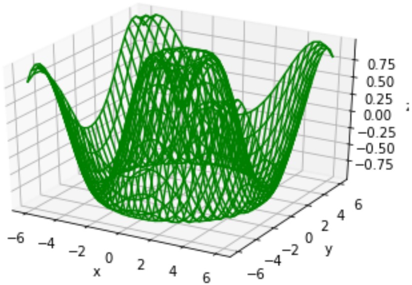
Plotting a surface (2)

The second step is to plot a wire-frame – this is our estimate of the surface.

```
fig = plt.figure()
ax = plt.axes(projection="3d")
ax.plot_wireframe(X, Y, Z, color='green')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

plt.show()
```

Plotting a surface (2)

The second step is to plot a wire-frame – this is our estimate of the surface.



Plotting a surface (3)

Finally, we'll project our surface onto our wire-frame estimate and extrapolate all of the points.

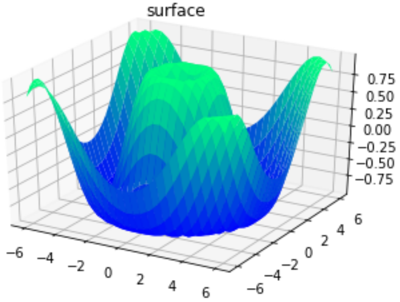
Plotting a surface (3)

Finally, we'll project our surface onto our wire-frame estimate and extrapolate all of the points.

```
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, cmap='winter')
ax.set_title('surface');
```

Plotting a surface (3)

Finally, we'll project our surface onto our wire-frame estimate and extrapolate all of the points.



Colormaps: https://matplotlib.org/3.1.1/tutorials/colors/colormaps.html

Draw the following surfaces:

$$\text{f} : \mathbb{R}^2 \to \mathbb{R} : (x, y) \to x^2 + y^2$$
$$\text{g} : \mathbb{R}^2 \to \mathbb{R}: (x, y) \to 4x - 2y + 3$$

Draw the following surfaces:

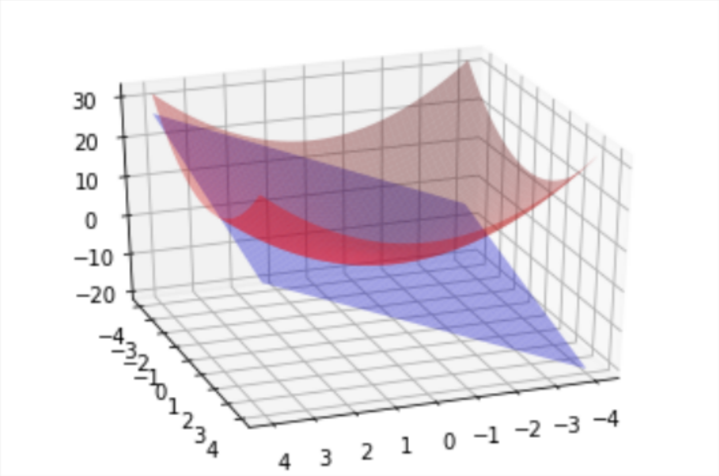
$$f : \mathbb{R}^2 \to \mathbb{R} : (x, y) \to x^2 + y^2$$
$$g : \mathbb{R}^2 \to \mathbb{R}: (x, y) \to 4x - 2y + 3$$

In [19]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
ax = plt.axes(projection = '3d')
x = np.arange(-4, 4, 0.01)
y = np.arange(-4, 4, 0.01)
X, Y = np.meshgrid(x, y)
def f(x,y):
    return x**2 + y**2
def g(x,y):
    return 4*x - 2*y + 3
Z1 = f(X, Y)
Z2 = g(X, Y)
ax.plot_surface(X, Y, Z1,color ='r')
ax.plot_surface(X, Y, Z2,  color ='b')
```

Draw the following surfaces:

$$f : \mathbb{R}^2 \to \mathbb{R} : (x, y) \to x^2 + y^2$$
$$g : \mathbb{R}^2 \to \mathbb{R}: (x, y) \to 4x - 2y + 3$$

>>>



Draw the following surface:

$$x = \sqrt{30}cos(t) + 3$$
$$y = \sqrt{30}sin(t) - 3$$
$$z = 6x - 6y + 12$$

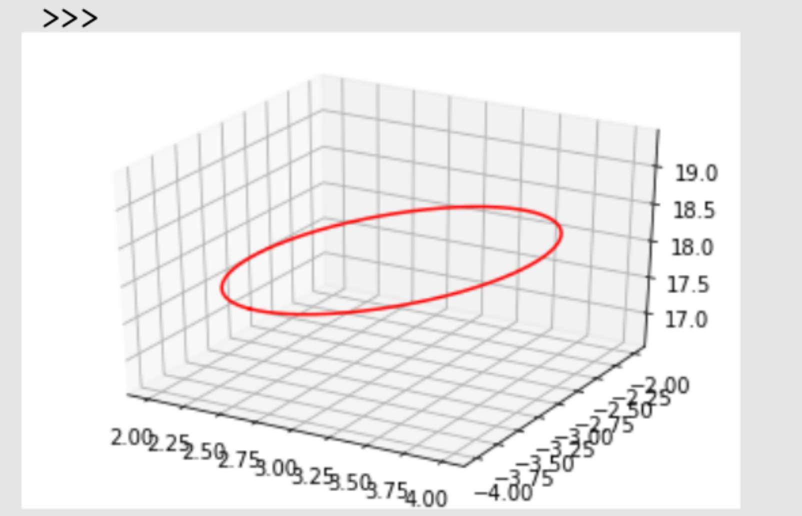
Draw the following surface:

$$x = \sqrt{30}cos(t) + 3$$
$$y = \sqrt{30}sin(t) - 3$$
$$z = 6x - 6y + 12$$

```
import numpy as np
import matplotlib.pyplot as plt

ax = plt.axes(projection = '3d')
t = np.arange(-np.pi,np.pi,0.01)
x_line = np.sqrt(30) * np.cos(t) + 3
y_line = np.sqrt(30) * np.sin(t) - 3
z_line = 6*x_line - 6*y_line + 12
ax.plot3D(x_line,y_line, z_line, 'r')
```

Draw the following surface:

$$x = \sqrt{30}cos(t) + 3$$
$$y = \sqrt{30}sin(t) - 3$$
$$z = 6x - 6y + 12$$



Example

plot the surface of the function $f(x, y)$ in the domain of $D(f) = [10, 10]^2 \subset \mathbb{R}$ from a good view. Where f is, $f : D(f)^2 \rightarrow \mathbb{R} : (x, y) \rightarrow e^x(x^2 - y^3)$

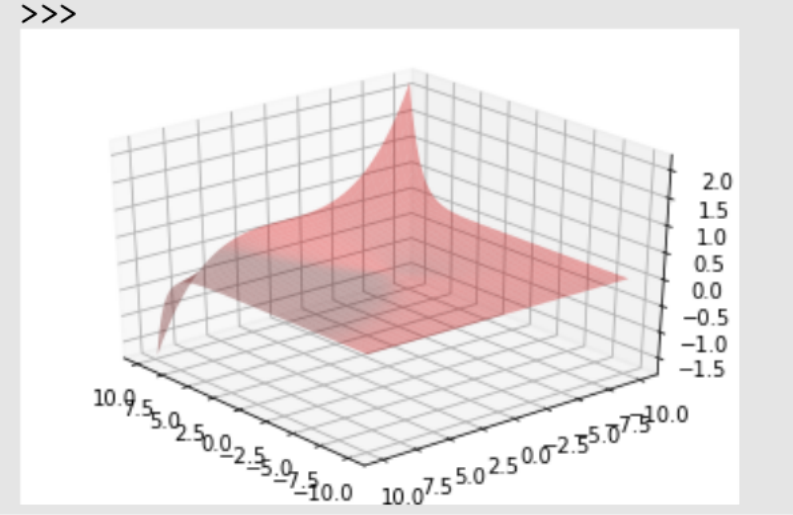
Example

plot the surface of the function $f(x, y)$ in the domain of $D(f) = [10, 10]^2 \subset \mathbb{R}$ from a good view. Where f is, $f : D(f)^2 \to \mathbb{R} : (x, y) \to e^x(x^2 - y^3)$

In [26]:
```
import numpy as np
import matplotlib.pyplot as plt
def f(x,y):
    return np.e**x * (x**2 - y**3)
x = np.arange(-10,10,0.1)
y = np.arange(-10,10,0.1)
X, Y = np.meshgrid(x,y)
Z = f(X, Y)
ax = plt.axes(projection = '3d')
ax.plot_surface(X, Y, Z, alpha = 0.35, color = 'r')
ax.view_init(30,140)
```

Example

plot the surface of the function $f(x,y)$ in the domain of $D(f) = [10,10]^2 \subset \mathbb{R}$ from a good view. Where f is, $f : D(f)^2 \to \mathbb{R} : (x,y) \to e^x(x^2 - y^3)$



1.Bissection Theorem

Bissection Theorem

The Bisection Method is a means of numerically approximating a solution to an equation: $f(x) = 0$

Bissection Theorem

The Bisection Method is a means of numerically approximating a solution to an equation: $f(x) = 0$
The fundamental mathematical principle underlying the Bisection Method is the Intermediate Value Theorem.

Theorem

Let $f : [a, b] \to [a, b]$ be a continuous function. Suppose that d is any value between $f(a)$ and $f(b)$. Then there is a c; $a < c < b$, such that $f(c) = d$.

The Intermediate Value Theorem implies that if $f(a)f(b) < 0$, then there is a point c; $a < c < b$ such that $f(c) = 0$.

Bisection Theorem

The Bisection Method is a means of numerically approximating a solution to an equation: $f(x) = 0$
The fundamental mathematical principle underlying the Bisection Method is the Intermediate Value Theorem.

Theorem

Let $f : [a, b] \to [a, b]$ be a continuous function. Suppose that d is any value between $f(a)$ and $f(b)$. Then there is a c; $a < c < b$, such that $f(c) = d$.

The Intermediate Value Theorem implies that if $f(a)f(b) < 0$, then there is a point c; $a < c < b$ such that $f(c) = 0$.

Thus if we have a continuous function f on an interval $[a, b]$ such that $f(a)f(b) < 0$, then $f(x) = 0$ has a solution in that interval.

Algorithm

The Intermediate Value Theorem not only guarantees a solution to the equation, but it also provides a means of numerically approximating a solution to arbitrary accuracy.

Algorithm

The Intermediate Value Theorem not only guarantees a solution to the equation, but it also provides a means of numerically approximating a solution to arbitrary accuracy.
Proceed as follows:

1. Let $\epsilon > 0$ be the upper bound for the error required of the answer

Algorithm

The Intermediate Value Theorem not only guarantees a solution to the equation, but it also provides a means of numerically approximating a solution to arbitrary accuracy.
Proceed as follows:

1. Let $\epsilon > 0$ be the upper bound for the error required of the answer
2. Compute $c = \frac{a+b}{2}$ and $d = f(c) \times f(a)$.

Algorithm

The Intermediate Value Theorem not only guarantees a solution to the equation, but it also provides a means of numerically approximating a solution to arbitrary accuracy.
Proceed as follows:

1. Let $\epsilon > 0$ be the upper bound for the error required of the answer
2. Compute $c = \frac{a+b}{2}$ and $d = f(c) \times f(a)$.
3. If $d < 0$, then let $b = c$ and $a = a$.

Algorithm

The Intermediate Value Theorem not only guarantees a solution to the equation, but it also provides a means of numerically approximating a solution to arbitrary accuracy.
Proceed as follows:

1. Let $\epsilon > 0$ be the upper bound for the error required of the answer
2. Compute $c = \frac{a+b}{2}$ and $d = f(c) \times f(a)$.
3. If $d < 0$, then let $b = c$ and $a = a$.
4. If $d > 0$, then let $a = c$ and $b = b$.

Algorithm

The Intermediate Value Theorem not only guarantees a solution to the equation, but it also provides a means of numerically approximating a solution to arbitrary accuracy.
Proceed as follows:

1. Let $\epsilon > 0$ be the upper bound for the error required of the answer
2. Compute $c = \frac{a+b}{2}$ and $d = f(c) \times f(a)$.
3. If $d < 0$, then let $b = c$ and $a = a$.
4. If $d > 0$, then let $a = c$ and $b = b$.
5. If $d = 0$, then c is a solution of $f(x) = 0$ and a solution has been found to the required accuracy.

Algorithm

The Intermediate Value Theorem not only guarantees a solution to the equation, but it also provides a means of numerically approximating a solution to arbitrary accuracy.
Proceed as follows:

1. Let $\epsilon > 0$ be the upper bound for the error required of the answer
2. Compute $c = \frac{a+b}{2}$ and $d = f(c) \times f(a)$.
3. If $d < 0$, then let $b = c$ and $a = a$.
4. If $d > 0$, then let $a = c$ and $b = b$.
5. If $d = 0$, then c is a solution of $f(x) = 0$ and a solution has been found to the required accuracy.
6. The new interval $[a, b]$ will then be half the length of the original $[a, b]$ and will contain a point $x \in [a, b]$ such that $f(x) = 0$.

Algorithm

The Intermediate Value Theorem not only guarantees a solution to the equation, but it also provides a means of numerically approximating a solution to arbitrary accuracy.
Proceed as follows:

1. Let $\epsilon > 0$ be the upper bound for the error required of the answer
2. Compute $c = \frac{a+b}{2}$ and $d = f(c) \times f(a)$.
3. If $d < 0$, then let $b = c$ and $a = a$.
4. If $d > 0$, then let $a = c$ and $b = b$.
5. If $d = 0$, then c is a solution of $f(x) = 0$ and a solution has been found to the required accuracy.
6. The new interval $[a, b]$ will then be half the length of the original $[a, b]$ and will contain a point $x \in [a, b]$ such that $f(x) = 0$.

Repeat 2) until either an exact solution is found in 5) or until at step 4) half the length of $[a, b]$ is less than, $\frac{b-a}{2} < 2$.

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$.

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

1st iteration :

Here $f(1) = -1 < 0$ and $f(2) = 5 > 0$

$\therefore$ Now, Root lies between 1 and 2

$$x_0 = \frac{1+2}{2} = 1.5$$

$$f(x_0) = f(1.5) = 0.875 > 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
$f(x)$	-1	-1	5

2^{nd} iteration :

Here $f(1) = -1 < 0$ and $f(1.5) = 0.875 > 0$

$\therefore$ Now, Root lies between 1 and 1.5

$$x_1 = \frac{1 + 1.5}{2} = 1.25$$

$$f(x_1) = f(1.25) = -0.29688 < 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

3^{rd} iteration :

Here $f(1.25) = -0.29688 < 0$ and $f(1.5) = 0.875 > 0$

$\therefore$ Now, Root lies between 1.25 and 1.5

$$x_2 = \frac{1.25 + 1.5}{2} = 1.375$$

$$f(x_2) = f(1.375) = 0.22461 > 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

4^{th} iteration :

Here $f(1.25) = -0.29688 < 0$ and $f(1.375) = 0.22461 > 0$

$\therefore$ Now, Root lies between 1.25 and 1.375

$$x_3 = \frac{1.25 + 1.375}{2} = 1.3125$$

$$f(x_3) = f(1.3125) = -0.05151 < 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
$f(x)$	-1	-1	5

5^{th} iteration :

Here $f(1.3125) = -0.05151 < 0$ and $f(1.375) = 0.22461 > 0$

$\therefore$ Now, Root lies between 1.3125 and 1.375

$$x_4 = \frac{1.3125 + 1.375}{2} = 1.34375$$

$$f(x_4) = f(1.34375) = 0.08261 > 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

6^{th} iteration :

Here $f(1.3125) = -0.05151 < 0$ and $f(1.34375) = 0.08261 > 0$

$\therefore$ Now, Root lies between 1.3125 and 1.34375

$$x_5 = \frac{1.3125 + 1.34375}{2} = 1.32812$$

$$f(x_5) = f(1.32812) = 0.01458 > 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

7^{th} iteration :

Here $f(1.3125) = -0.05151 < 0$ and $f(1.32812) = 0.01458 > 0$

$\therefore$ Now, Root lies between 1.3125 and 1.32812

$$x_6 = \frac{1.3125 + 1.32812}{2} = 1.32031$$

$$f(x_6) = f(1.32031) = -0.01871 < 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

8^{th} iteration :

Here $f(1.32031) = -0.01871 < 0$ and $f(1.32812) = 0.01458 > 0$

$\therefore$ Now, Root lies between 1.32031 and 1.32812

$$x_7 = \frac{1.32031 + 1.32812}{2} = 1.32422$$

$$f(x_7) = f(1.32422) = -0.00213 < 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

9^{th} iteration :

Here $f(1.32422) = -0.00213 < 0$ and $f(1.32812) = 0.01458 > 0$

$\therefore$ Now, Root lies between 1.32422 and 1.32812

$$x_8 = \frac{1.32422 + 1.32812}{2} = 1.32617$$

$$f(x_8) = f(1.32617) = 0.00621 > 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

10^{th} iteration :

Here $f(1.32422) = -0.00213 < 0$ and $f(1.32617) = 0.00621 > 0$

$\therefore$ Now, Root lies between 1.32422 and 1.32617

$$x_9 = \frac{1.32422 + 1.32617}{2} = 1.3252$$

$$f(x_9) = f(1.3252) = 0.00204 > 0$$

Video+Example1

Find the approximate intersection between $h(x) = x^3$ and $g(x) = x + 1$.

Solution

Find the intersection between $h(x)$ and $g(x)$ means to find an x_0 such that $h(x_0) = g(x_0)$ i.e $h(x_0) - g(x_0) = 0$. This means that x_0 is a root of $h(x) - g(x)$. So let us find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method.

Here

x	0	1	2
f(x)	-1	-1	5

11^{th} iteration :

Here $f(1.32422) = -0.00213 < 0$ and $f(1.3252) = 0.00204 > 0$

$\therefore$ Now, Root lies between 1.32422 and 1.3252

$$x_{10} = \frac{1.32422 + 1.3252}{2} = 1.32471$$

$$f(x_{10}) = f(1.32471) = -0.00005 < 0$$

Approximate root of the equation $x^3 - x - 1 = 0$ using Bisection mehtod is 1.32471

n	a	$f(a)$	b	$f(b)$	$c = \frac{a+b}{2}$	$f(c)$	Update
1	1	-1	2	5	1.5	0.875	$b = c$
2	1	-1	1.5	0.875	1.25	-0.29688	$a = c$
3	1.25	-0.29688	1.5	0.875	1.375	0.22461	$b = c$
4	1.25	-0.29688	1.375	0.22461	1.3125	-0.05151	$a = c$
5	1.3125	-0.05151	1.375	0.22461	1.34375	0.08261	$b = c$
6	1.3125	-0.05151	1.34375	0.08261	1.32812	0.01458	$b = c$
7	1.3125	-0.05151	1.32812	0.01458	1.32031	-0.01871	$a = c$
8	1.32031	-0.01871	1.32812	0.01458	1.32422	-0.00213	$a = c$
9	1.32422	-0.00213	1.32812	0.01458	1.32617	0.00621	$b = c$
10	1.32422	-0.00213	1.32617	0.00621	1.3252	0.00204	$b = c$
11	1.32422	-0.00213	1.3252	0.00204	1.32471	-0.00005	$a = c$

Example 2

write a script that finds the approximate intersection between f and g, through use of the bisection theorem. Additionally, graph $f(x)$ and $g(x)$ and their intersection.

$$f : \mathbb{R} \to \mathbb{R} : x \to cos^2(x), g : \mathbb{R} \to \mathbb{R} : x \to x^2$$

Solution

1. Start by defining the function $i(x)$, whose root is going to be the value of intersection required.
2. Define the function that is going to use the bissection method and has 4 inputs: i, a, b, ϵ, where a and b are the boundaries of your interval of study and ϵ is the smallest value of accepted error.
3. With the IF test, check if we have a root in this interval or not.
4. If not, keep repeating the bissection method as long as $|a-b|/2 > \epsilon$
5. Stop the test when the image of the midpoint is equal to 0.
6. Draw the two functions with their intersection point.

Solution

```
import numpy as np

def i(x):
        return (np.cos(x))**2 - x**2
```

Solution

```
def bisection_method(f,a,b,epsilon):
        if f(a)*f(b) > 0:
                return 'no root exists'
```

Solution

```
def bisection_method(f,a,b,epsilon):
        if f(a)*f(b) > 0:
                return 'no root exists'
        else:
                while np.absolute(a-b)/2 > epsilon:
                        midpoint = (a+b)/2
```

Solution

```
def bisection_method(f,a,b,epsilon):
        if f(a)*f(b) > 0:
                return 'no root exists'
        else:
                while np.absolute(a-b)/2 > epsilon:
                        midpoint = (a+b)/2
                        if f(midpoint) == 0:
                                return midpoint
```

Solution

```
def bisection_method(f,a,b,epsilon):
        if f(a)*f(b) > 0:
                return 'no root exists'
        else:
                while np.absolute(a-b)/2 > epsilon:
                        midpoint = (a+b)/2
                        if f(midpoint) == 0:
                                return midpoint
                        elif f(midpoint)*f(a) < 0:
                                b= midpoint
```

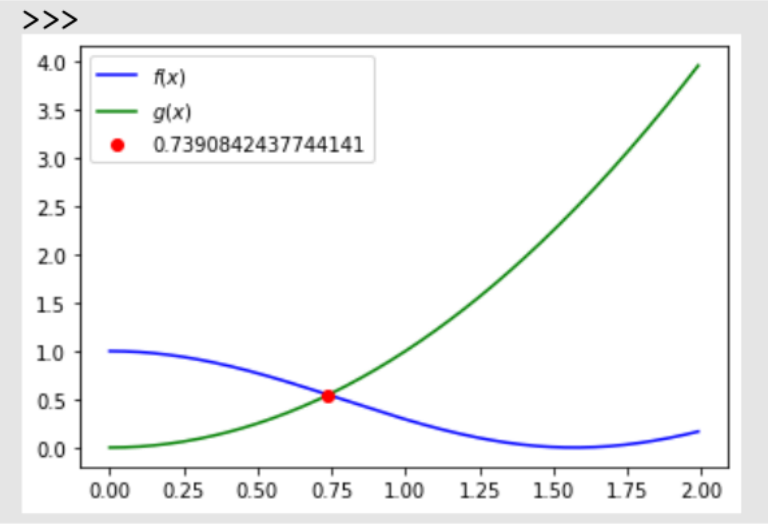
Solution

```
def bisection_method(f,a,b,epsilon):
        if f(a)*f(b) > 0:
                return 'no root exists'
        else:
                while np.absolute(a-b)/2 > epsilon:
                        midpoint = (a+b)/2
                        if f(midpoint) == 0:
                                return midpoint
                        elif f(midpoint)*f(a) < 0:
                                b= midpoint
                        else:
                                a = midpoint
                return midpoint
```

Solution

```
import numpy as np

def i(x):
        return (np.cos(x))**2 - x**2

def bisection_method(f,a,b,epsilon):
        if f(a)*f(b) > 0:
                return 'no root exists'
        else:
                while np.absolute(a-b)/2 > epsilon:
                        midpoint = (a+b)/2
                        if f(midpoint) == 0:
                                return midpoint
                        elif f(midpoint)*f(a) < 0:
                                b= midpoint
                        else:
                                a = midpoint
                return midpoint

def f(x):
        return (np.cos(x))**2
def g(x):
        return x**2

x = np.arange(0,2,0.01)
intercept = bisection_method(i,0,2,10**-6)
plt.plot(x, f(x), 'b', label = '$f(x)$')
plt.plot(x, g(x), 'g', label = '$g(x)$')
plt.plot(intercept, f(intercept), 'or', label =
str(intercept))
plt.legend()
```

Solution



>>>
f(x)
g(x)
0.7390842437744141
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

1. Widgets on Jupiter

What are widgets?

What are widgets?

Widgets are eventful python objects that have a representation in the browser, often as a control like a slider, textbox, etc.

What are widgets?

Widgets are eventful python objects that have a representation in the browser, often as a control like a slider, textbox, etc.

What can they be used for?

What are widgets?

Widgets are eventful python objects that have a representation in the browser, often as a control like a slider, textbox, etc.

What can they be used for?

You can use widgets to build interactive GUIsfor your notebooks. You can also use widgets to synchronizestateful and stateless information between Python and JavaScript.

Using widgets

Baby steps.

Using widgets

Baby steps.

1. To use the widget framework, you need to import *ipywidgets*:

Using widgets

Baby steps.

1. To use the widget framework, you need to import *ipywidgets*:

```
[1]: import ipywidgets as widgets
```

Using widgets

Baby steps.

1. To use the widget framework, you need to import *ipywidgets*:

```
[1]: import ipywidgets as widgets
```

2. Widgets have their own display *representation* which allows them to be displayed using IPython's display framework. Constructing and returning an IntSlider automatically displays the widget.

Using widgets

Baby steps.

1. To use the widget framework, you need to import *ipywidgets*:

```
[1]: import ipywidgets as widgets
```

2. Widgets have their own display *representation* which allows them to be displayed using IPython's display framework. Constructing and returning an IntSlider automatically displays the widget.

```
[2]: widgets.IntSlider()
```

[2]:

Using widgets

Baby steps.

1. To use the widget framework, you need to import *ipywidgets*:

```
[1]: import ipywidgets as widgets
```

2. Widgets have their own display *representation* which allows them to be displayed using IPython's display framework. Constructing and returning an IntSlider automatically displays the widget.

```
[2]: widgets.IntSlider()
```

[2]:

3. You can also explicitly display the widget using *display(...)*.

Using widgets

Baby steps.

1. To use the widget framework, you need to import *ipywidgets*:

```
[1]: import ipywidgets as widgets
```

2. Widgets have their own display *representation* which allows them to be displayed using IPython's display framework. Constructing and returning an IntSlider automatically displays the widget.

```
[2]: widgets.IntSlider()
```

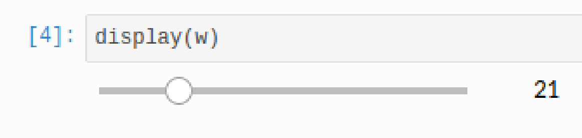
[2]:

3. You can also explicitly display the widget using *display(...)*.

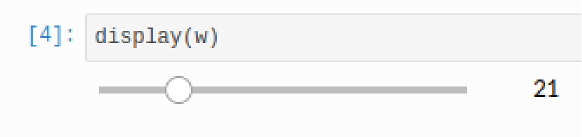
```
[3]: from IPython.display import display
     w = widgets.IntSlider()
     display(w)
```

4. If you display the same widget twice, the displayed instances in the front-end will remain in sync with each other. Try dragging the slider **below** and watch the slider **above**.

4. If you display the same widget twice, the displayed instances in the front-end will remain in sync with each other. Try dragging the slider **below** and watch the slider **above**.

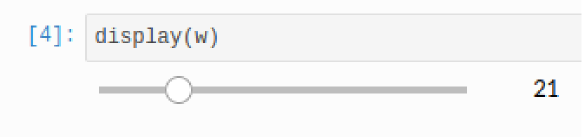


4. If you display the same widget twice, the displayed instances in the front-end will remain in sync with each other. Try dragging the slider **below** and watch the slider **above**.



5. You can close a widget by calling its close() method.

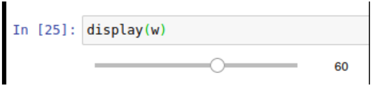
4. If you display the same widget twice, the displayed instances in the front-end will remain in sync with each other. Try dragging the slider **below** and watch the slider **above**.



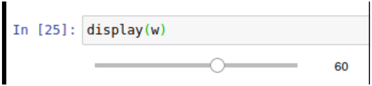
5. You can close a widget by calling its close() method.

6. All of the IPython widgets share a similar naming scheme. To read the value of a widget, you can query its value property.

6. All of the IPython widgets share a similar naming scheme. To read the value of a widget, you can query its value property.



6. All of the IPython widgets share a similar naming scheme. To read the value of a widget, you can query its value property.



7. In addition to value , most widgets sharekeys, description, and disabled. To see the entire list of synchronized, stateful properties of any specific **widget**, you can query the keys property.

6. All of the IPython widgets share a similar naming scheme. To read the value of a widget, you can query its value property.

```
In [25]: display(w)
```

```
In [26]: w.value
Out[26]: 60
```

7. In addition to value , most widgets sharekeys, description, and disabled. To see the entire list of synchronized, stateful properties of any specific **widget**, you can query the keys property.

```
[10]: w.keys

[10]: ['_dom_classes',
       '_model_module',
       '_model_module_version',
       '_model_name',
       '_view_count',
       '_view_module',
       '_view_module_version',
       '_view_name',
       'continuous_update',
       'description',
       'disabled',
       'layout',
       'max',
       'min',
       'orientation',
       'readout',
       'readout_format',
       'step',
       'style',
       'tabbable',
       'tooltip',
       'value']
```

8. While creating a widget, you can set some or all of the initial values of that widget by defining them as keyword arguments in the widgetâĂŹs constructor

8. While creating a widget, you can set some or all of the initial values of that widget by defining them as keyword arguments in the widgetâĂŹs constructor

```
[11]: widgets.Text(value='Hello World!', disabled=True)

[11]: Hello World!
```

8. While creating a widget, you can set some or all of the initial values of that widget by defining them as keyword arguments in the widgetâĂŹs constructor

```
[11]: widgets.Text(value='Hello World!', disabled=True)
```

```
[11]: Hello World!
```

9. If you need to display the same value two different ways, youâĂŹll have to use two different widgets. Instead of attempting to manually synchronize the values of the two widgets, you can use the *link* or *jslink* function to **link** two properties together.

8. While creating a widget, you can set some or all of the initial values of that widget by defining them as keyword arguments in the widgetâĂŹs constructor

```
[11]: widgets.Text(value='Hello World!', disabled=True)
```

[11]:

9. If you need to display the same value two different ways, youâĂŹll have to use two different widgets. Instead of attempting to manually synchronize the values of the two widgets, you can use the *link* or *jslink* function to **link** two properties together.

```
[12]: a = widgets.FloatText()
      b = widgets.FloatSlider()
      display(a,b)

      mylink = widgets.jslink((a, 'value'), (b, 'value'))
```

8. While creating a widget, you can set some or all of the initial values of that widget by defining them as keyword arguments in the widgetâĂŹs constructor

```
[11]: widgets.Text(value='Hello World!', disabled=True)
```

[11]:

9. If you need to display the same value two different ways, youâĂŹll have to use two different widgets. Instead of attempting to manually synchronize the values of the two widgets, you can use the *link* or *jslink* function to **link** two properties together.

```
[12]: a = widgets.FloatText()
      b = widgets.FloatSlider()
      display(a,b)

      mylink = widgets.jslink((a, 'value'), (b, 'value'))
```

10. Unlinking the widgets is simple. All you have to do is call .unlink on the link object.

8. While creating a widget, you can set some or all of the initial values of that widget by defining them as keyword arguments in the widgetâĂŹs constructor

```
[11]: widgets.Text(value='Hello World!', disabled=True)
```

[11]: Hello World!

9. If you need to display the same value two different ways, youâĂŹll have to use two different widgets. Instead of attempting to manually synchronize the values of the two widgets, you can use the *link* or *jslink* function to **link** two properties together.

```
[12]: a = widgets.FloatText()
      b = widgets.FloatSlider()
      display(a,b)

      mylink = widgets.jslink((a, 'value'), (b, 'value'))
```

26.9

26.90

10. Unlinking the widgets is simple. All you have to do is call .unlink on the link object.

```
[13]: # mylink.unlink()
```

Numeric widgets

There are many widgets distributed with ipywidgets that are designed to display numeric values. Widgets exist for displaying integers and floats, both bounded and unbounded. The integer widgets share a similar naming scheme to their floating point counterparts. By replacing Float with Int in the widget name, you can find the Integer equivalent.

11. The slider is displayed with a specified, initial value. Lower and upper bounds are defined by min and max, and the value can be incremented according to the step parameter.

11. The slider is displayed with a specified, initial value. Lower and upper bounds are defined by min and max, and the value can be incremented according to the step parameter.
12. The slider's label is defined by description parameter.

11. The slider is displayed with a specified, initial value. Lower and upper bounds are defined by min and max, and the value can be incremented according to the step parameter.
12. The slider's label is defined by description parameter.
13. The slider's orientation is either 'horizontal' (default) or 'vertical'.

11. The slider is displayed with a specified, initial value. Lower and upper bounds are defined by min and max, and the value can be incremented according to the step parameter.
12. The slider's label is defined by description parameter.
13. The slider's orientation is either 'horizontal' (default) or 'vertical'.
14. readout displays the current value of the slider next to it. The options are True (default) or False.

11. The slider is displayed with a specified, initial value. Lower and upper bounds are defined by min and max, and the value can be incremented according to the step parameter.
12. The slider's label is defined by description parameter.
13. The slider's orientation is either 'horizontal' (default) or 'vertical'.
14. readout displays the current value of the slider next to it. The options are True (default) or False.
15. readout_format specifies the format function used to represent slider value. The default is $'.2f'$(2 digit float). 'd' is used for integers.

11. The slider is displayed with a specified, initial value. Lower and upper bounds are defined by min and max, and the value can be incremented according to the step parameter.
12. The slider's label is defined by description parameter.
13. The slider's orientation is either 'horizontal' (default) or 'vertical'.
14. readout displays the current value of the slider next to it. The options are True (default) or False.
15. readout_format specifies the format function used to represent slider value. The default is '.2f'(2 digit float). 'd' is used for integers.

```
[2]: widgets.IntSlider(
         value=7,
         min=0,
         max=10,
         step=1,
         description='Test:',
         disabled=False,
         continuous_update=False,
         orientation='horizontal',
         readout=True,
         readout_format='d'
     )
```

16. The FloatLogSlider has a log scale, which makes it easy to have a slider that covers a wide range of positive magnitudes. The min and max refer to the minimum and maximum exponents of the base, and the value refers to the actual value of the slider.

16. The FloatLogSlider has a log scale, which makes it easy to have a slider that covers a wide range of positive magnitudes. The min and max refer to the minimum and maximum exponents of the base, and the value refers to the actual value of the slider.

```
[5]: widgets.FloatLogSlider(
         value=10,
         base=10,
         min=-10, # max exponent of base
         max=10, # min exponent of base
         step=0.2, # exponent step
         description='Log Slider'
     )
```

16. The FloatLogSlider has a log scale, which makes it easy to have a slider that covers a wide range of positive magnitudes. The min and max refer to the minimum and maximum exponents of the base, and the value refers to the actual value of the slider.

```
[5]: widgets.FloatLogSlider(
         value=10,
         base=10,
         min=-10, # max exponent of base
         max=10, # min exponent of base
         step=0.2, # exponent step
         description='Log Slider'
     )
```

17. IntRangeSlider and FloatRangeSlide.

16. The FloatLogSlider has a log scale, which makes it easy to have a slider that covers a wide range of positive magnitudes. The min and max refer to the minimum and maximum exponents of the base, and the value refers to the actual value of the slider.

```
[5]: widgets.FloatLogSlider(
         value=10,
         base=10,
         min=-10, # max exponent of base
         max=10, # min exponent of base
         step=0.2, # exponent step
         description='Log Slider'
     )
```

17. IntRangeSlider and FloatRangeSlide.

```
[6]: widgets.IntRangeSlider(
         value=[5, 7],
         min=0,
         max=10,
         step=1,
         description='Test:',
         disabled=False,
         continuous_update=False,
         orientation='horizontal',
         readout=True,
         readout_format='d',
     )
```

```
[7]: widgets.FloatRangeSlider(
         value=[5, 7.5],
         min=0,
         max=10.0,
         step=0.1,
         description='Test:',
         disabled=False,
         continuous_update=False,
         orientation='horizontal',
         readout=True,
         readout_format='.1f',
     )
```

Outline

Session XV

1. Test 1 + Correction

Questions+ Solutions

Question	Assume the following list definition: `a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']` Which display correct output?
Answer	✔ >>> print(a[4::-2]) ['quux', 'baz', 'foo']
	>>> print(a[-6]) Traceback (most recent call last): File "<stdin>", line 1, in <module> IndexError: list index out of range
	✔ >>> print(a[-5:-3]) ['bar', 'baz']
	>>> a[:] is a True

Question	List a is defined as follows: a = ['a', 'b', 'c'] Which of the following statements adds 'd' and 'e' to the end of a, so that it then equals ['a', 'b', 'c', 'd', 'e']:
Answer	✓ a.extend(['d', 'e']) a[-1:] = ['d', 'e'] a.append(['d', 'e']) ✓ a[len(a):] = ['d', 'e'] a += 'de' ✓ a += ['d', 'e']

Question

Suppose s is assigned as follows:

s = 'foobar'
All of the following expressions produce the same result except one. Which one?

Answer

- s[::-1][::-5]
- ✔ s[::-5]
- s[::5]
- s[::-1][-1] + s[len(s)-1]
- s[0] + s[-1]

Short Answer: You have a list a defined as follo...

Question

You have a list a defined as follows:

a = [1, 2, 7, 8]
Write a Python statement using **slice assignment** that will fill in the missing values so that a equals [1, 2, 3, 4, 5, 6, 7, 8].

Answer

a[2:2] = [3, 4, 5, 6]

Question	Consider this statement:

```
>>> print('foo\\bar\nbaz')
```

Which of the following is the correct output?

Answer

- foo\bar\nbaz
- ✔ foo\bar
  baz
- foo\\barnbaz
- foo\\bar\nbaz

True/False: Every time when we modify the string,...

Question: Every time when we modify the string, Python Always create a new String and assign a new string to that variable

Answer

- ✔ True
- False

True/False: In Python 3, the maximum value for an...

Question: In Python 3, the maximum value for an integer is 2^{63} - 1:

Answer

- True
- ✔ False

Question	What is the output? for i in range(10, 15, 1): print(i)
Answer	✓ 10, 11, 12, 13, 14 10, 11, 12, 13, 14, 15 9,10, 11, 12, 13, 14

1. Multiple Choice: What is the output of this code? i=s...

Question

What is the output of this code?

```
i=s=0

while i<=3:
    s+=i
    i=i+1

print(s)
```

Answer

- 3
- 4
- ✓ 6
- 0

Question	What is the output?

```
a=10
if a<5:
    a=20
elif a>1:
    a=500
elif a>100:
    a=1
else:
    a=0

print(a)
```

Answer

- 0
- 1
- 10
- 20
- ✓ 500
- None of the above

Question	What is the output?

```
def f(L):
    L.append(50)
    L.append(30)
    return

M=[1,2,3]
print(M)
f(M)
print(M)
```

Answer

[1,2,3,50,30]
[1,2,3,50,30]

[1,2,3]
[1,2,3]

✔ [1,2,3]
[1,2,3,50,30]

[1,2,3,50,30]
[1,2,3]

Question

What is the output?

```
def f(a):
    a+=5
    print(a)
    return a

a=6
print(a)
f(a)
print(a)
```

Answer

Error

✔ 6 11 6

6 11 11

6 Followed by an error

6 11 6 6

6 11 11 11

Question	In Python, a variable may be assigned a value of one type, and then later assigned a value of a different type:
Answer	✔ True False

Multiple Answer: List a is defined as follows: a =...

Question	List a is defined as follows: a = [1, 2, 3, 4, 5] Select all of the following statements that remove the middle element 3 from a so that it equals [1, 2, 4, 5]:
Answer	a[2] = [ ] ✔ a[2:3] = [ ] ✔ a.remove(3) a[2:2] = [ ] ✔ del a[2]

Question	What is the output?

```
def myst(a,b):
    a=a+b
    b=a-b
    a=a-b
    return [a,b]
```

Answer

- [3,5]
- [5,3]
- [6,-2]
- ✔ None of the above

'. Multiple Choice: What is the output? i=0while i<3:...

Question	What is the output?

```
i=0
while i<3:
    print( i)
    i++
    print(i+1)
```

Answer

- 0 2 1 3 2 4
- 1 0 2 4 3 5
- 0 1 2 3 4 5
- ✔ Error

Question	What is the result of the following statement: `list(([a,b,c,d,e] + 'fghi')[3:6])`
Answer	['d', 'e', 'f']
	'def'
	✔ It raises an error
	[100, 101, 102]
	['d', 'e', 'f']

9. **Short Answer: What is the slice expression that giv...**

Question	What is the slice expression that gives every third character of string s, starting with the last character and proceeding backward to the first?
Answer	s[::-3]

Question	Which of the following are true of Python lists?
Answer	A list may contain any type of object except another list
	All elements in a list must be of the same type
	✔ A given object may appear in a list more than once
	✔ There is no conceptual limit to the size of a list
	These represent the same list: ['a', 'b', 'c'] ['c', 'a', 'b']

. Multiple Answer: Which of the following are true:

Question	Which of the following are true:
Answer	✔ s[:] == s
	✔ s[::-1][::-1] == s
	s[::-1][::-1] is s
	s[:] is s

Question	Which of the following are valid ways to specify the string literal **foo'bar** in Python:
Answer	✔ 'foo\'bar'
	'foo'bar'
	✔ """foo'bar"""
	✔ "foo'bar"
	'foo''bar'

. Multiple Choice: Which of the following statements ass...

Question	Which of the following statements assigns the value 100 to the variable x in Python:
Answer	let x = 100
	x ← 100
	✔ x = 100
	x << 100
	x := 100

Question	def calculate (num1, num2): res = num1 * num2 return(res) calculate(5, 6)
Answer	20 ✔ 30 The program executed with errors

5. Multiple Choice: for x in range(0.5, 5.5, 0.5): &...

Question	for x in range(0.5, 5.5, 0.5): print(x)
Answer	[0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5] [0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5] ✔ The Program executed with errors

Question	sampleList = ["Jon", "Kelly", "Jessa"] sampleList.append(2,"Scott") print(sampleList)
Answer	['Jon', 'Kelly', 'Scott', 'Jessa'] ['Jon', 'Kelly', 'Jessa', 'Scott'] ['Jon', 'Scott', 'Kelly', 'Jessa'] ✔ The program executed with errors

. Multiple Choice: var = "James" * 2 * 3 &nbs...

Question	var = "James" * 2 * 3 print(var)
Answer	✔ JamesJamesJamesJamesJamesJames JamesJamesJamesJamesJames Error: invalid syntax

Question	var1 = 1 var2 = 2 var3 = "3" print(var + var2 + var3)
Answer	6 ✔ Error. Mixing operators between numbers and strings are not supported 33 123

9. Multiple Choice: var="James Bond" print(var[2::-1])

Question	var="James Bond" print(var[2::-1])
Answer	dnoB semaJ ✔ maJ dno Jam

Question	x = 36 / 4 * (3 + 2) * 4 + 2 print(x)
Answer	✔ 182.0 37 117 The Program executed with errors

Question What is the output?

```
def op1(a, b, c):
    return a + b * c
def op2(a, b, c):
    return a * 2 + b
def op3(a, b, c):
    return c * 3 - b
i = 1
j = 1
k = 1
if i > 5:
    result = op1(i,j,k)
elif j < 3:
    result = op2(j, k, i)
else:
    result = op3(k, j, i)
print (result)
```

Answer

- ✔ 3
- 5
- 2
- 1

1. Exercises

Time format

Given a time in **12-hour AM/PM format**, convert it to **military** (24-hour) time.

Time format

Given a time in **12-hour AM/PM format**, convert it to **military** (24-hour) time.

Note

Midnight is 12:00:00 AM on a 12-hour clock and 00:00:00 on a 24-hour clock.
Noon is 12:00:00 PM on 12-hour clock and 12:00:00 on 24-hour clock.

Time format

Given a time in **12-hour AM/PM format**, convert it to **military** (24-hour) time.

Note

Midnight is 12:00:00 AM on a 12-hour clock and 00:00:00 on a 24-hour clock.
Noon is 12:00:00 PM on 12-hour clock and 12:00:00 on 24-hour clock.

Examples :

```
Input : 11:21:30 PM
Output : 23:21:30

Input : 12:12:20 AM
Output : 00:12:20
```

Solution

```python
# Python program to convert time
# from 12 hour to 24 hour format

# Function to convert the date format
def convert24(str1):

    # Checking if last two elements of time
    # is AM and first two elements are 12
    if str1[-2:] == "AM" and str1[:2] == "12":
        return "00" + str1[2:-2]

    # remove the AM
    elif str1[-2:] == "AM":
        return str1[:-2]

    # Checking if last two elements of time
    # is PM and first two elements are 12
    elif str1[-2:] == "PM" and str1[:2] == "12":
        return str1[:-2]

    else:

        # add 12 to hours and remove PM
        return str(int(str1[:2]) + 12) + str1[2:8]
```

Second way

Given a time in **military (24-hour)** , convert it to **12-hour AM/PM format**time.

Second way

Given a time in **military (24-hour)** , convert it to **12-hour AM/PM format**time.

Note

Midnight is 00:00:00 on a 24-hour clock and 12:00:00 AM on a 12-hour clock.
Noon is 12:00:00 on 24-hour clock and 12:00:00 PM on 12-hour clock.

Second way

Given a time in **military (24-hour)** , convert it to **12-hour AM/PM format**time.

Note

Midnight is 00:00:00 on a 24-hour clock and 12:00:00 AM on a 12-hour clock.
Noon is 12:00:00 on 24-hour clock and 12:00:00 PM on 12-hour clock.

Examples:

```
Input : 17:35:20
Output : 5:35:20 PM

Input : 00:10:24
Output : 12:10:24 AM
```

Solution

```python
# Convert Function which takes in
# 24hour time and convert it to
# 12 hour format
def convert12(str):

    # Get Hours
    h1 = ord(str[0]) - ord('0');
    h2 = ord(str[1]) - ord('0');

    hh = h1 * 10 + h2;

    # Finding out the Meridien of time
    # ie. AM or PM
    Meridien="";
    if (hh < 12):
        Meridien = "AM";
    else:
        Meridien = "PM";

    hh %= 12;

    # Handle 00 and 12 case separately
    if (hh == 0):
        print("12", end = "");

        # Printing minutes and seconds
        for i in range(2, 8):
            print(str[i], end = "");

    else:
        print(hh,end="");

        # Printing minutes and seconds
        for i in range(2, 8):
            print(str[i], end = "");

    # After time is printed
    # cout Meridien
    print(" " + Meridien);
```

Is prime?

Given a positive integer, check if the number is prime or not.

Note

A prime is a natural number greater than 1 that has no positive divisors other than 1 and itself. Examples of first few prime numbers are $\{2, 3, 5, \ldots$

Is prime?

Given a positive integer, check if the number is prime or not.

Note

A prime is a natural number greater than 1 that has no positive divisors other than 1 and itself. Examples of first few prime numbers are $\{2, 3, 5, \ldots$

Examples:

```
Input:  n = 11
Output: true

Input:  n = 15
Output: false

Input:  n = 1
Output: false
```

Is prime?

Given a positive integer, check if the number is prime or not.

Note

A prime is a natural number greater than 1 that has no positive divisors other than 1 and itself. Examples of first few prime numbers are $\{2, 3, 5, \ldots$

```python
def isPrime(n):

    # Corner case
    if n <= 1:
        return False

    # Check from 2 to n-1
    for i in range(2, n):
        if n % i == 0:
            return False;

    return True

# Driver Program to test above function
print("true") if isPrime(11) else print("false")
print("true") if isPrime(14) else print("false")
```

Find largest prime factor of a number

Given a positive integer n; $(1 \leq n \leq 10^{15})$. Find the largest prime factor of a number. .

Find largest prime factor of a number

Given a positive integer n; $(1 \leq n \leq 10^{15})$. Find the largest prime factor of a number. .

```
Input: 6
Output: 3
Explanation
Prime factor of 6 are- 2, 3
Largest of them is \'3\'

Input: 15
Output: 5
```

Find largest prime factor of a number

Given a positive integer n; $(1 \leq n \leq 10^{15})$. Find the largest prime factor of a number.

```
def maxPrimeFactors(n):
    l=[]
    for i in range(2,n):
        if i%n==0:
            l.append(i)
    p=[]
    for i in l:
        if isPrime(i)==True:
            p.append(i)
    return max(p)
```

2. Fibbonacci

Golden number

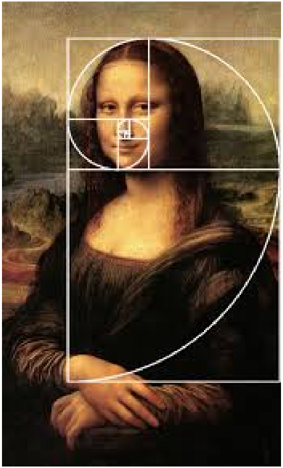
What do we have by Tool in common with:

Golden number

What do we have by Tool in common with: sunflowers,

Golden number

What do we have by Tool in common with: sunflowers, the Golden ratio,

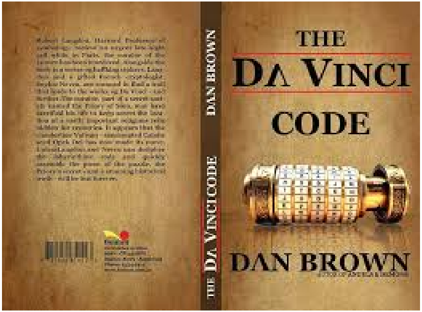


Golden number

What do we have by Tool in common with: sunflowers, the Golden ratio, fur tree cones,

Golden number

What do we have by Tool in common with: sunflowers, the Golden ratio, fur tree cones, The Da Vinci Code



Golden number

What do we have by Tool in common with: sunflowers, the Golden ratio, fur tree cones, The Da Vinci Code the song "Lateralus".

"Lateralus"

(1)Black,
(1) then,
(2) white are,
(3) all I see,
(5) in my infancy,
(8) red and yellow then came to be,
(5) reaching out to me,
(3) lets me see.
(2) There is,
(1) so,

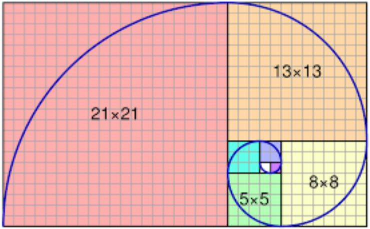
(1) much,
(2) more that
(3) beckons me,
(5) to look through to these,
(8) infinite possibilities.
(13) As below so above and beyond I imagine,
(8) drawn outside the lines of reason.
(5) Push the envelope.
(3) Watch it bend.

5.5 Fibonacci's Rabbits 48

Golden number

What do we have by Tool in common with: sunflowers, the Golden ratio, fur tree cones, The Da Vinci Code the song "Lateralus".

The Fibonacci numbers are the numbers of the following sequence of integer values: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$



Golden number

What do we have by Tool in common with: sunflowers, the Golden ratio, fur tree cones, The Da Vinci Code the song "Lateralus".
The Fibonacci numbers are the numbers of the following sequence of integer values: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$The Fibonacci numbers are defined by:

$$F_n = F_{n-1} + F_{n-2} \ with \ F_0 = 0 \ and \ F_1 = 1$$

Golden number

What do we have by Tool in common with: sunflowers, the Golden ratio, fur tree cones, The Da Vinci Code the song "Lateralus".
The Fibonacci numbers are the numbers of the following sequence of integer values: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$ The Fibonacci numbers are defined by:

$$F_n = F_{n-1} + F_{n-2} \text{ with } F_0 = 0 \text{ and } F_1 = 1$$

History

The Fibonacci sequence is named after the mathematician Leonardo of Pisa, who is better known as Fibonacci. In his book "Liber Abaci" (published 1202) he introduced the sequence as an exercise dealing with bunnies. His sequence of the Fibonacci numbers begins with $F_1 = 1$, while in modern mathematics the sequence starts with $F_0 = 0$. But this has no effect on the other members of the sequence.

Coding Fibonnaci

Write in two different ways, the functions $fib(n)$ and $fibi(n)$. Where the first is written in a reccursive way and the second one with a for Loop.

Coding Fibonnaci

Write in two different ways, the functions *fib*(n) and *fibi*(n). Where the first is written in a reccursive way and the second one with a for Loop.

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Coding Fibonnaci

Write in two different ways, the functions *fib*(n) and *fibi*(n). Where the first is written in a reccursive way and the second one with a for Loop.

```
def fibi(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a
```

Time comparaison

If you check the functions fib() and fibi(), you will find out that the iterative version fibi()is a lot faster than the recursive version fib(). To get an idea of how much this **"a lot faster"** can be, we have written a script where we you the timeit module to measure the calls:

Time comparaison

If you check the functions fib() and fibi(), you will find out that the iterative version fibi()is a lot faster than the recursive version fib(). To get an idea of how much this **"a lot faster"** can be, we have written a script where we you the timeit module to measure the calls:

```
from timeit import Timer
from fibo import fib

t1 = Timer("fib(10)","from fibo import fib")

for i in range(1,41):
        s = "fib(" + str(i) + ")"
        t1 = Timer(s,"from fibo import fib")
        time1 = t1.timeit(3)
        s = "fibi(" + str(i) + ")"
        t2 = Timer(s,"from fibo import fibi")
        time2 = t2.timeit(3)
        print("n=%2d, fib: %8.6f, fibi:  %7.6f, percent: %10.2f" % (i, time1, time2, time1/time2))
```

Change fibo by __*main*__.

Time comparaison

If you check the functions fib() and fibi(), you will find out that the iterative version fibi()is a lot faster than the recursive version fib(). To get an idea of how much this **"a lot faster"** can be, we have written a script where we you the timeit module to measure the calls: Change fibo by __*main*__.

```
n= 1, fib: 0.000004, fibi:  0.000005, percent:       0.81
n= 2, fib: 0.000005, fibi:  0.000005, percent:       1.00
n= 3, fib: 0.000006, fibi:  0.000006, percent:       1.00
n= 4, fib: 0.000008, fibi:  0.000005, percent:       1.62
n= 5, fib: 0.000013, fibi:  0.000006, percent:       2.20
n= 6, fib: 0.000020, fibi:  0.000006, percent:       3.36
n= 7, fib: 0.000030, fibi:  0.000006, percent:       5.04
n= 8, fib: 0.000047, fibi:  0.000008, percent:       5.79
n= 9, fib: 0.000075, fibi:  0.000007, percent:      10.50
n=10, fib: 0.000118, fibi:  0.000007, percent:      16.50
n=11, fib: 0.000198, fibi:  0.000007, percent:      27.70
n=12, fib: 0.000287, fibi:  0.000007, percent:      41.52
n=13, fib: 0.000480, fibi:  0.000007, percent:      69.45
n=14, fib: 0.000780, fibi:  0.000007, percent:     112.83
n=15, fib: 0.001279, fibi:  0.000008, percent:     162.55
n=16, fib: 0.002059, fibi:  0.000009, percent:     233.41
n=17, fib: 0.003439, fibi:  0.000011, percent:     313.59
n=18, fib: 0.005794, fibi:  0.000012, percent:     486.04
n=19, fib: 0.009219, fibi:  0.000011, percent:     840.59
n=20, fib: 0.014366, fibi:  0.000011, percent:    1309.89
n=21, fib: 0.023137, fibi:  0.000013, percent:    1764.42
n=22, fib: 0.036963, fibi:  0.000013, percent:    2818.80
n=23, fib: 0.060626, fibi:  0.000012, percent:    4985.96
n=24, fib: 0.097643, fibi:  0.000013, percent:    7584.17
n=25, fib: 0.157224, fibi:  0.000013, percent:   11989.91
n=26, fib: 0.253764, fibi:  0.000013, percent:   19352.05
n=27, fib: 0.411353, fibi:  0.000012, percent:   34506.80
n=28, fib: 0.673918, fibi:  0.000014, percent:   47908.76
n=29, fib: 1.086484, fibi:  0.000015, percent:   72334.03
n=30, fib: 1.742688, fibi:  0.000014, percent:  123887.51
n=31, fib: 2.861763, fibi:  0.000014, percent:  203442.44
n=32, fib: 4.648224, fibi:  0.000015, percent:  309461.33
n=33, fib: 7.339578, fibi:  0.000014, percent:  521769.86
n=34, fib: 11.980462, fibi:  0.000014, percent:  851689.83
n=35, fib: 19.426206, fibi:  0.000016, percent: 1216110.64
n=36, fib: 30.840097, fibi:  0.000015, percent: 2053218.13
```

1. Widgets

Using widgets

Baby steps.

1. To use the widget framework, you need to import *ipywidgets*:

```
[1]: import ipywidgets as widgets
```

2. Widgets have their own display *representation* which allows them to be displayed using IPython's display framework. Constructing and returning an IntSlider automatically displays the widget.

```
[2]: widgets.IntSlider()
```

[2]:

3. You can also explicitly display the widget using *display(...)*.

```
[3]: from IPython.display import display
     w = widgets.IntSlider()
     display(w)
```

4. The FloatLogSlider has a log scale, which makes it easy to have a slider that covers a wide range of positive magnitudes. The min and max refer to the minimum and maximum exponents of the base, and the value refers to the actual value of the slider.

```
[5]: widgets.FloatLogSlider(
         value=10,
         base=10,
         min=-10, # max exponent of base
         max=10, # min exponent of base
         step=0.2, # exponent step
         description='Log Slider'
     )
```

5. IntRangeSlider and FloatRangeSlide.

```
[6]: widgets.IntRangeSlider(
         value=[5, 7],
         min=0,
         max=10,
         step=1,
         description='Test:',
         disabled=False,
         continuous_update=False,
         orientation='horizontal',
         readout=True,
         readout_format='d',
     )
```

```
[7]: widgets.FloatRangeSlider(
         value=[5, 7.5],
         min=0,
         max=10.0,
         step=0.1,
         description='Test:',
         disabled=False,
         continuous_update=False,
         orientation='horizontal',
         readout=True,
         readout_format='.1f',
     )
```

6. IntProgress and FloatProgress.

6. IntProgress and FloatProgress.

```
[8]: widgets.IntProgress(
         value=7,
         min=0,
         max=10,
         step=1,
         description='Loading:',
         bar_style='', # 'success', 'info', 'warning', 'danger' or ''
         orientation='horizontal'
     )
```

```
[9]: widgets.FloatProgress(
         value=7.5,
         min=0,
         max=10.0,
         step=0.1,
         description='Loading:',
         bar_style='info',
         orientation='horizontal'
     )
```

7. BoundedIntText and BoundedFloatText.

7. BoundedIntText and BoundedFloatText.

```
[10]: widgets.BoundedIntText(
          value=7,
          min=0,
          max=10,
          step=1,
          description='Text:',
          disabled=False
      )
```

```
[11]: widgets.BoundedFloatText(
          value=7.5,
          min=0,
          max=10.0,
          step=0.1,
          description='Text:',
          disabled=False
      )
```

Boolean widgets

There are three widgets that are designed to display a boolean value.

Boolean widgets

There are three widgets that are designed to display a boolean value.

8. ToggleButton .

Boolean widgets

There are three widgets that are designed to display a boolean value.

8. ToggleButton .

```
[14]: widgets.ToggleButton(
          value=False,
          description='Click me',
          disabled=False,
          button_style='', # 'success', 'info', 'warning', 'danger' or ''
          tooltip='Description',
          icon='check' # (FontAwesome names without the `fa-` prefix)
      )
```

Boolean widgets

There are three widgets that are designed to display a boolean value.

8. ToggleButton .

```
[14]: widgets.ToggleButton(
          value=False,
          description='Click me',
          disabled=False,
          button_style='', # 'success', 'info', 'warning', 'danger' or ''
          tooltip='Description',
          icon='check' # (FontAwesome names without the `fa-` prefix)
      )
```

9. Valid .

   The valid widget provides a read-only indicator.

Boolean widgets

There are three widgets that are designed to display a boolean value.

8. ToggleButton .

```
[14]: widgets.ToggleButton(
          value=False,
          description='Click me',
          disabled=False,
          button_style='', # 'success', 'info', 'warning', 'danger' or ''
          tooltip='Description',
          icon='check' # (FontAwesome names without the `fa-` prefix)
      )
```

9. Valid .
   The valid widget provides a read-only indicator.

```
[16]: widgets.Valid(
          value=False,
          description='Valid!',
      )
```

10. Checkbox.

10. Checkbox.

- value specifies the value of the checkbox
- indent parameter places an indented checkbox, aligned with other controls. Options are True (default) or False.

```
[15]: widgets.Checkbox(
          value=False,
          description='Check me',
          disabled=False,
          indent=False
      )
```

Selection widget

Widgets used to display a selection list.

11. Dropdown.

Selection widget

Widgets used to display a selection list.

11. Dropdown.

```
[17]: widgets.Dropdown(
          options=['1', '2', '3'],
          value='2',
          description='Number:',
          disabled=False,
      )
```

Selection widget

Widgets used to display a selection list.

11. Dropdown.

```
[17]: widgets.Dropdown(
          options=['1', '2', '3'],
          value='2',
          description='Number:',
          disabled=False,
      )
```

```
[18]: widgets.Dropdown(
          options=[('One', 1), ('Two', 2), ('Three', 3)],
          value=2,
          description='Number:',
      )
```

12. RadioButtons.

12. RadioButtons.

```
[19]: widgets.RadioButtons(
          options=['pepperoni', 'pineapple', 'anchovies'],
      #    value='pineapple', # Defaults to 'pineapple'
      #    layout={'width': 'max-content'}, # If the items' names are long
          description='Pizza topping:',
          disabled=False
      )
```

12. RadioButtons.

```
[19]: widgets.RadioButtons(
          options=['pepperoni', 'pineapple', 'anchovies'],
      #    value='pineapple', # Defaults to 'pineapple'
      #    layout={'width': 'max-content'}, # If the items' names are long
          description='Pizza topping:',
          disabled=False
      )
```

13. SelectMultiple.

12 RadioButtons.

```
[19]: widgets.RadioButtons(
          options=['pepperoni', 'pineapple', 'anchovies'],
      #    value='pineapple', # Defaults to 'pineapple'
      #    layout={'width': 'max-content'}, # If the items' names are long
          description='Pizza topping:',
          disabled=False
      )
```

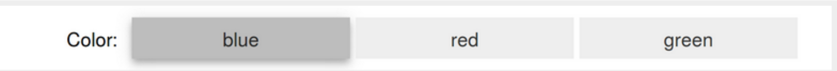
13 SelectMultiple.
Multiple values can be selected with shift and/or ctrl (or command) pressed and mouse clicks or arrow keys.

```
[25]: widgets.SelectMultiple(
          options=['Apples', 'Oranges', 'Pears'],
          value=['Oranges'],
          #rows=10,
          description='Fruits',
          disabled=False
      )
```

14. Horizontal Buttons.

14 Horizontal Buttons.

```
color_buttons = widgets.ToggleButtons(
    options=['blue', 'red', 'green'],
    description='Color:',
)
color_buttons
```



Password format

The Password widget hides user input on the screen.

Password format

The Password widget hides user input on the screen.
This widget is not a secure way to collect sensitive information because:

Password format

The Password widget hides user input on the screen.
This widget is not a secure way to collect sensitive information because:

- The contents of the Password widget are transmitted unencrypted.

Password format

The Password widget hides user input on the screen.
This widget is not a secure way to collect sensitive information because:

- The contents of the Password widget are transmitted unencrypted.
- If the widget state is saved in the notebook the contents of the Password widget is stored as plain text.

```
[29]: widgets.Password(
          value='password',
          placeholder='Enter password',
          description='Password:',
          disabled=False
      )
```

Date and Color picker

The date picker widget works in Chrome, Firefox and IE Edge, but does not currently work in Safari.

Date and Color picker

The date picker widget works in Chrome, Firefox and IE Edge, but does not currently work in Safari.

```
[36]: widgets.DatePicker(
          description='Pick a Date',
          disabled=False
      )
```

Date and Color picker

The date picker widget works in Chrome, Firefox and IE Edge, but does not currently work in Safari.

```
[36]: widgets.DatePicker(
          description='Pick a Date',
          disabled=False
      )
```

```
[37]: widgets.ColorPicker(
          concise=False,
          description='Pick a color',
          value='blue',
          disabled=False
      )
```

Play animation widget

The Play widget is useful to perform animations by iterating on a sequence of integers with a certain speed. The value of the slider below is linked to the player.

Play animation widget

The Play widget is useful to perform animations by iterating on a sequence of integers with a certain speed. The value of the slider below is linked to the player.

```
[35]: play = widgets.Play(
          value=50,
          min=0,
          max=100,
          step=1,
          interval=500,
          description="Press play",
          disabled=False
      )
      slider = widgets.IntSlider()
      widgets.jslink((play, 'value'), (slider, 'value'))
      widgets.HBox([play, slider])
```

2. Interactive section

Interact

The interact function (ipywidgets.interact) automatically creates user interface controls for exploring code and data interactively.

Interact

The interact function (ipywidgets.interact) automatically creates user interface controls for exploring code and data interactively.

```
[1]: from __future__ import print_function
     from ipywidgets import interact, interactive, fixed, interact_manual
     import ipywidgets as widgets
```

Interact

The interact function (ipywidgets.interact) automatically creates user interface controls for exploring code and data interactively.

```
[1]: from __future__ import print_function
     from ipywidgets import interact, interactive, fixed, interact_manual
     import ipywidgets as widgets
```

To use interact, you need to define a function that you want to explore.

Interact

The interact function (ipywidgets.interact) automatically creates user interface controls for exploring code and data interactively.

```
[1]: from __future__ import print_function
     from ipywidgets import interact, interactive, fixed, interact_manual
     import ipywidgets as widgets
```

To use interact, you need to define a function that you want to explore.

```
In [17]: def f(x):
             return 4*x-3
         interact(f,x=2);
```

x 2

5

Several arguments

In [17]:
```python
def f(x):
    return 4*x-3
interact(f,x=2);
```

x 2

5

In [18]:
```python
interact(f, x=widgets.IntSlider(min=-10, max=30, step=1, value=10));
```

x 24

93

In [19]:
```python
interact(f, x=(0,8,2));
```

x 4

13

In [21]:
```python
interact(f, x=[('one', 10), ('two', 20)]);
```

x one

37

Interactive function

In addition to interact, IPython provides another function, interactive, that is useful when you want to reuse the widgets that are produced or access the data that is bound to the controls.

Interactive function

In addition to interact, IPython provides another function, interactive, that is useful when you want to reuse the widgets that are produced or access the data that is bound to the controls.

```
[18]: from IPython.display import display
      def f(a, b):
          display(a + b)
          return a+b
```

```
[19]: w = interactive(f, a=10, b=20)
      30
```

```
[22]: display(w)
```

Interactive_output

interactive_output provides additional flexibility: you can control how the elements are laid out.

Interactive_output

interactive_output provides additional flexibility: you can control how the elements are laid out.

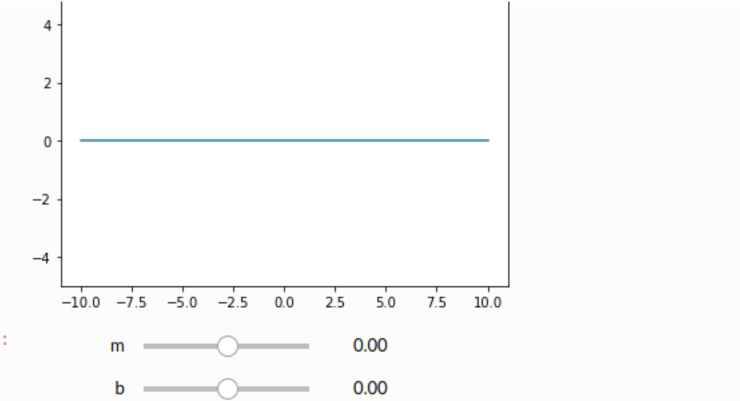
```
[31]: a = widgets.IntSlider()
      b = widgets.IntSlider()
      c = widgets.IntSlider()
      ui = widgets.HBox([a, b, c])
      def f(a, b, c):
          print((a, b, c))

      out = widgets.interactive_output(f, {'a': a, 'b': b, 'c': c})

      display(ui, out)
```

```
(0, 0, 0)
```

Example

```
[33]: %matplotlib inline
from ipywidgets import interactive
import matplotlib.pyplot as plt
import numpy as np

def f(m, b):
    plt.figure(2)
    x = np.linspace(-10, 10, num=1000)
    plt.plot(x, m * x + b)
    plt.ylim(-5, 5)
    plt.show()

interactive_plot = interactive(f, m=(-2.0, 2.0), b=(-3, 3, 0.5))
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot
```

Example



Interactive functions

Here is an example joining the widgets with matplotlib;

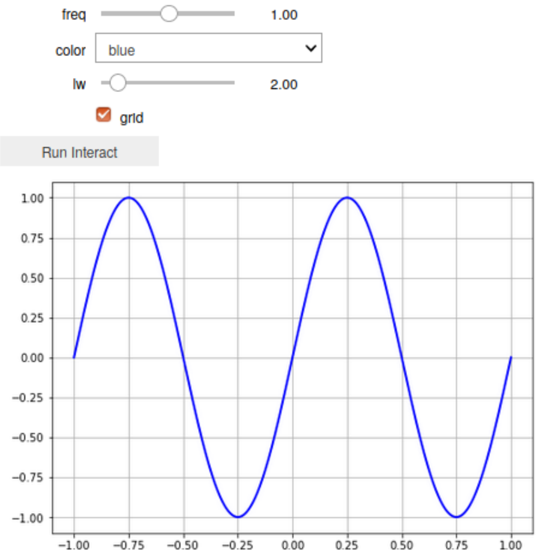
Interactive functions

Here is an example joining the widgets with matplotlib;

```
In [26]: import numpy as np
         import matplotlib.pyplot as plt
         @widgets.interact_manual(
             color=['blue', 'red', 'green'], lw=(1., 10.))
         def plot(freq=1., color='blue', lw=2, grid=True):
             t = np.linspace(-1., +1., 1000)
             fig, ax = plt.subplots(1, 1, figsize=(8, 6))
             ax.plot(t, np.sin(2 * np.pi * freq * t),
                     lw=lw, color=color)
             ax.grid(grid)
```

Interactive functions

Here is an example joining the widgets with matplotlib;



3. More exercises

Perfect numbers

In number theory, a perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself.

Perfect numbers

In number theory, a perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself.

Example

The first perfect number is 6, because 1, 2, and 3 are its proper positive divisors, and $1 + 2 + 3 = 6$. The next perfect number is $28 = 1 + 2 + 4 + 7 + 14$. This is followed by the perfect numbers 496 and 8128.

Perfect numbers

In number theory, a perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself.

Example

The first perfect number is 6, because 1, 2, and 3 are its proper positive divisors, and $1 + 2 + 3 = 6$. The next perfect number is $28 = 1 + 2 + 4 + 7 + 14$. This is followed by the perfect numbers 496 and 8128.

```python
def perfect_number(n):
    sum = 0
    for x in range(1, n):
        if n % x == 0:
            sum += x
    return sum == n
print(perfect_number(6))
```

Palindrome

A palindrome is nothing but any number or a string which remains unaltered when reversed.

Palindrome

A palindrome is nothing but any number or a string which remains unaltered when reversed.

Example: 12321

Output: Yes, a Palindrome number

Example: RACECAR

Output: Yes, a Palindrome string

Palindrome

A palindrome is nothing but any number or a string which remains unaltered when reversed.

```
string=input(("Enter a string:"))
if(string==string[::-1]):
        print("The string is a palindrome")
else:
        print("Not a palindrome")
```

Given an array of positive integers. All numbers occur even number of times except one number which occurs odd number of times.

Given an array of positive integers. All numbers occur even number of times except one number which occurs odd number of times.

Examples :

```
Input : arr = {1, 2, 3, 2, 3, 1, 3}
Output : 3

Input : arr = {5, 7, 2, 7, 5, 2, 5}
Output : 5
```

Given an array of positive integers. All numbers occur even number of times except one number which occurs odd number of times.

```
# function to find the element occurring odd
# number of times
def getOddOccurrence(arr, arr_size):

    for i in range(0, arr_size):
        count = 0
        for j in range(0, arr_size):
            if arr[i] == arr[j]:
                count+= 1

        if (count % 2 != 0):
            return arr[i]

    return -1
```