

Object Oriented Programming

based on Python 3

AUTM08017

Introduction to Python3



Dr Diarmuid Ó Briain

Version 2.0 [15 September 2024]

Copyright © 2024 C²S Consulting

Licensed under the EUPL, Version 1.2 or – as soon they will be approved by the European Commission - subsequent versions of the EUPL (the "Licence");

You may not use this work except in compliance with the Licence.

You may obtain a copy of the Licence at:

https://joinup.ec.europa.eu/sites/default/files/custom-page/attachment/eupl_v1.2_en.pdf

Unless required by applicable law or agreed to in writing, software distributed under the Licence is distributed on an "AS IS" basis, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the Licence for the specific language governing permissions and limitations under the Licence.

Dr Diarmuid Ó Briain



```
~$ python3
>>> import sys
>>> sys.version
'3.12.3 (main, Jul 31 2024, 17:43:48) [GCC 13.2.0]'
```

Table of Contents

1. Tutorial: Flowcharts and Pseudocode.....	5
1.1 Rules For Creating Flowchart.....	6
1.2 Pseudocode.....	6
1.3 Using a Flowchart to understand a program requirement.....	6
2. Exercise 1.1.....	8
3. Brief History of Python.....	9
4. Python home.....	10
5. Installing Python.....	11
5.1 Python on GNU/Linux.....	11
5.2 Python on Microsoft Windows.....	12
6. Running Python.....	13
6.1 The Python Interpreter on GNU/Linux - python3.....	13
6.2 The Python Interpreters on Microsoft Windows, python.....	14
6.3 iPython.....	14
6.4 bPython.....	15
7. Integrated Development and Learning Environment (IDLE).....	16
7.1 Text editors and Integrated Development Environment (IDE).....	18
7.1.1 CudaText.....	18
7.1.2 Sublime text.....	19
7.1.3 Notepad++ on Microsoft Windows.....	19
8. Running Programs.....	20
9. Mathematical Operators.....	24
10. Simple functions.....	25
11. Getting help on functions within a module.....	26
12. Bytecode files.....	26
13. Exercise #1.2.....	27

Table of Figures

Illustration 1: Basic Symbols used in Flowchart Designs.....	5
Illustration 2: Flowchart, Pseudocode and code for Hello World.....	6
Illustration 3: Flowchart for simple add program.....	7
Illustration 4: Another Flowchart to Python Code example.....	8
Illustration 5: Python documentation.....	10
Illustration 6: Python for Microsoft Windows.....	12
Illustration 7: Python installed on Microsoft Windows.....	12
Illustration 8: IDLE.....	16
Illustration 9: IDLE Editor and Run module option.....	16
Illustration 10: hello_world.py in gedit.....	17
Illustration 11: Gnome Gedit.....	18
Illustration 12: CudaText.....	18
Illustration 13: Sublime text.....	19
Illustration 14: Notepad++.....	19
Illustration 15: factorial.py flowchart.....	20
Illustration 16: factorial.py ran on GNU/Linux.....	21
Illustration 17: factorial.py loops through function.....	21
Illustration 18: factorial.py ran on Microsoft Windows.....	22
Illustration 19: RE-script.py.....	22
Illustration 20: unique_sorted_list.py.....	23
Illustration 21: Simple functions.....	25
Illustration 22: Python Virtual Machine.....	26

1. Tutorial: Flowcharts and Pseudocode

Flowcharts can be useful for planning programs. The flowchart is a form of graphical representation of a program algorithm. The typical flowchart uses symbols which are connected among them to indicate the flow of information and processing. The process of drawing a flowchart for an algorithm is known as “flowcharting”.

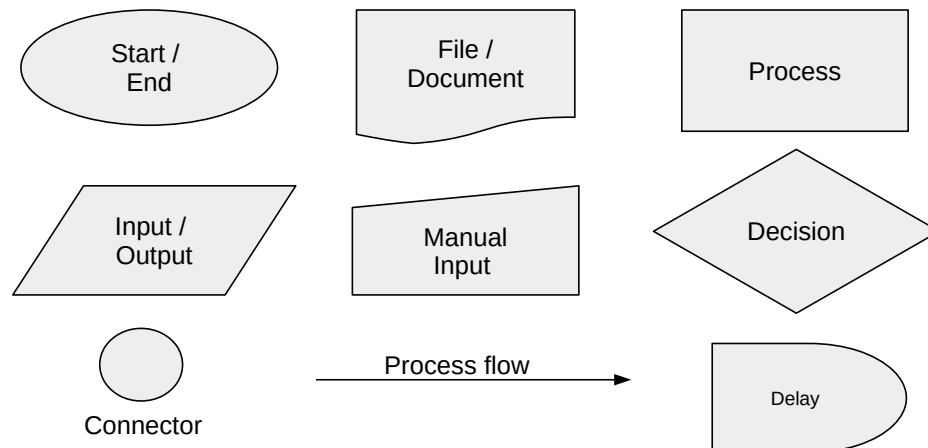


Illustration 1: Basic Symbols used in Flowchart Designs

As displayed in Illustration 1 each symbol has a particular function:

- **Terminal:** The oval symbol indicates **Start**, **Stop** and **Halt** in a program's logic flow. A pause/halt is generally used in a program logic under some error conditions. Terminal is the first and last symbols in the flowchart.
- **Input/Output:** A parallelogram denotes any function of input/output type. Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.
- **Connectors:** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.
- **File/Document:** Programs often read in input or write output to a file or document.
- **Manual Input:** Users maybe prompted for input from the program, from a Graphical User Interface (GUI) or from a webpage.
- **Process flow:** Process flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.
- **Processing:** A box represents arithmetic instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.
- **Decision Diamond:** represents a decision point. Decision based operations such as **yes/no** question or **true/false** are indicated by diamond in flowchart.
- **Delay:** It is often advantageous to include delays in the program processing.

1.1 Rules For Creating Flowchart

A flowchart is a graphical representation of an algorithm, it should therefore follow some rules:

- Rule 1: Flowchart opening statement must be '**start**' keyword.
- Rule 2: Flowchart ending statement must be '**end**' keyword.
- Rule 3: All symbols in the flowchart must be connected with an **arrow** line.

1.2 Pseudocode

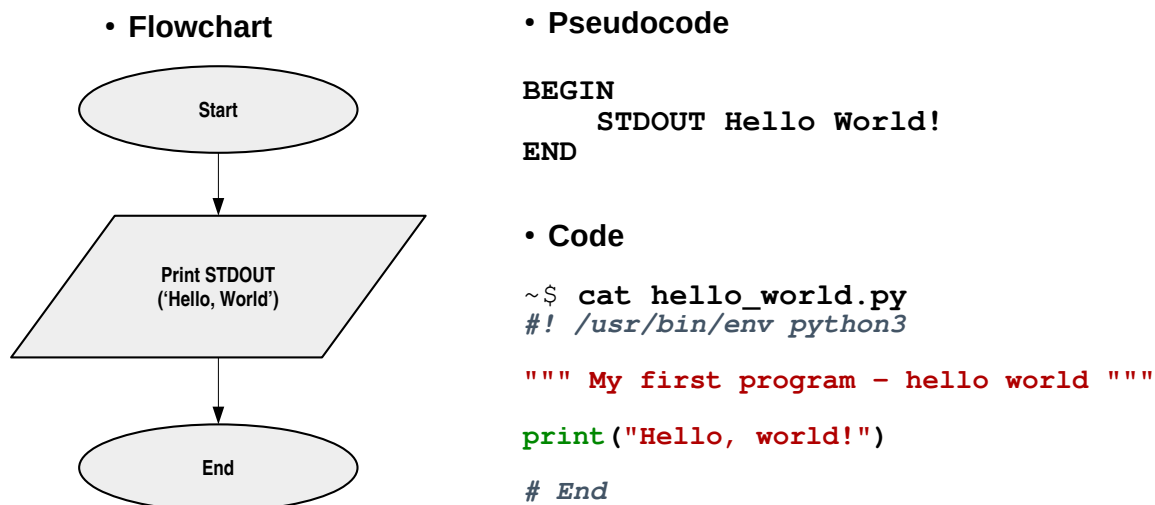


Illustration 2: Flowchart, Pseudocode and code for Hello World

Pseudocode is a plain language description of the steps in an algorithm or another system. Pseudocode often uses structural conventions of a normal programming language, but is intended for human reading rather than machine reading.

Consider Illustration 2, The Flowchart and Pseudocode could apply to any of the “Hello World” programs considered in the Introduction to Programming topic; however, the code illustrated is specific to the Python3 programming language.

1.3 Using a Flowchart to understand a program requirement

Consider Illustration 3, this is the flowchart for a program to be developed. Consider the steps:

1. Program starts.
2. Next, the program asks for a number.
3. 10 is added to the number.
4. Next, the resulting sum is printed.
5. Finally, the program ends.

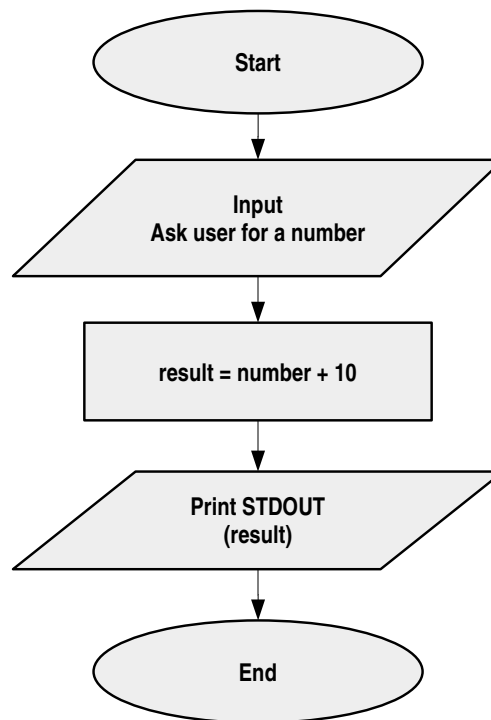


Illustration 3: Flowchart for simple add program

Now building a python program to meet the Flowchart requirements.

```
~$ cat simple_add.py
#!/usr/bin/env python3

""" Simple add """

# // Start //

num = 10
int_ = int(input("Enter a number: "))
ans = int_ + num
print (f"{int_} + {num} = {ans}")

# // End //
```

```
~$ chmod +x simple_add.py
~$ ./simple_add.py
Enter an number: 3
3 + 10 = 13
```

Consider another example in Illustration 4.

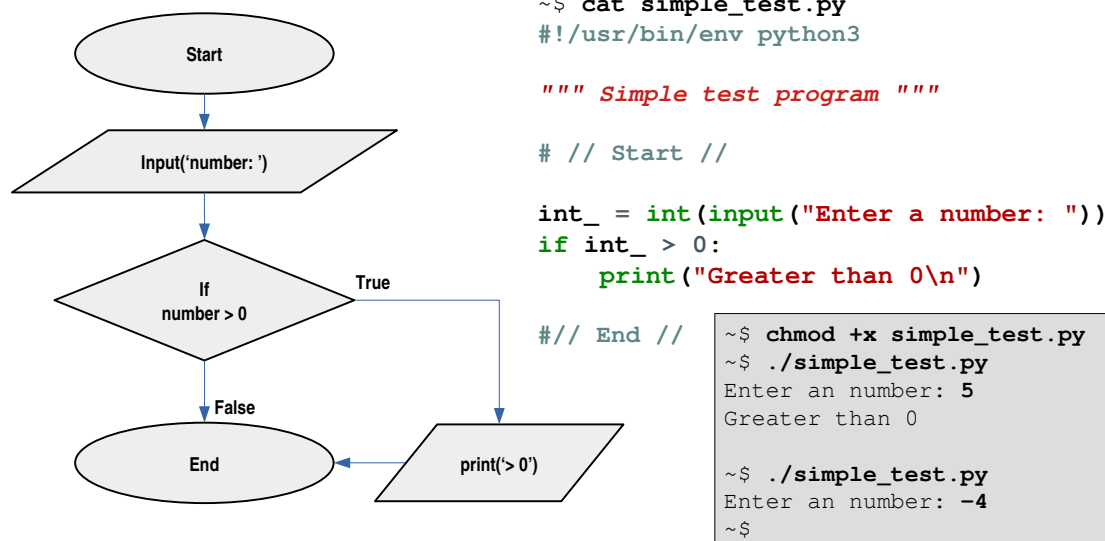


Illustration 4: Another Flowchart to Python Code example

2. Exercise 1.1

Draw a flowchart for the following program, and write the associated program.

- The program starts.
- Next, the program asks a user for a number.
- If the number is greater than zero, the program prints "Greater than 0".
- If the number is less than one, the program prints "Less than 1".
- Then the program prints "Done" and the program ends.

3. Brief History of Python

The history of the Python programming language dates back to the late 1980s and its implementation was started in December 1989 by **Guido van Rossum** at Centrum Wiskunde & Informatica (CWI) which is the Dutch National Research Institute for Mathematics and Computer Science. It was named after Monty Python, a British comedy from the 1960s. Guido van Rossum is Python's principal author, and has continued in a central role in deciding the direction of Python. He had the Python community title of Benevolent Dictator for Life (BDFL), however; he stepped down from the position in July 2018.

Python has been Open Source from inception and while it is considered a scripting language, it is in fact much more. It is scalable, object oriented and functional and has been used by Google.

Python 2.0 was released on October 16, 2000 and a further major change was the introduction of Python 3.0 on December 3, 2008. This is a major, backwards-incompatible release. Many of Python3 features have been backported to the backwards-compatible Python 2.6 and 2.7.



“Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered.”

- Guido van Rossum

4. Python home

Python information can be found at <http://docs.python.org/>, the site also includes a good Python tutorial.

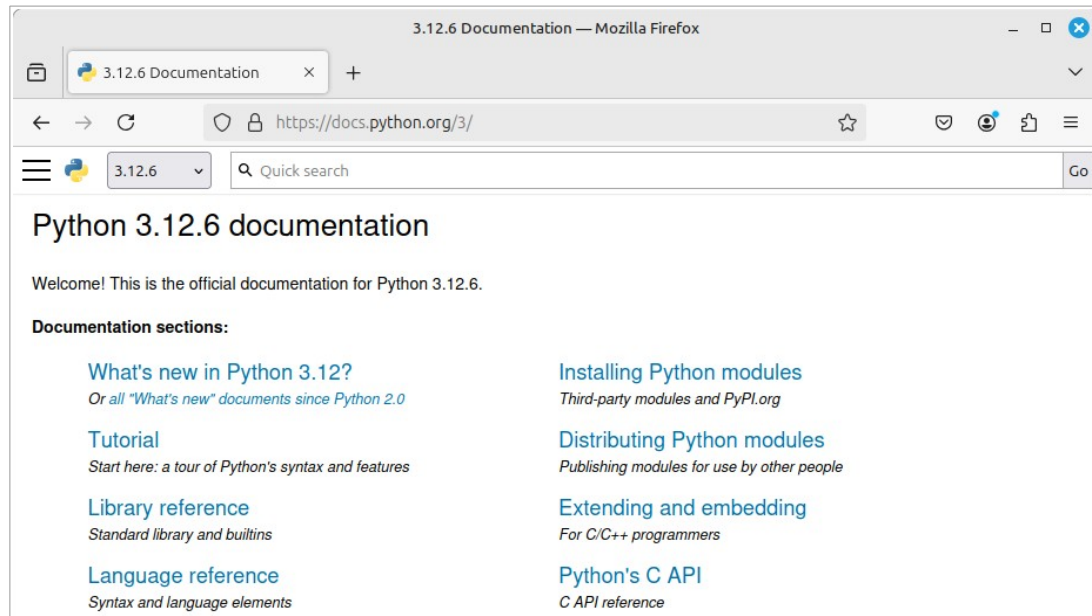


Illustration 5: Python documentation

5. Installing Python

5.1 Python on GNU/Linux

Python comes pre-installed on most GNU/Linux, UNIX and MAC OS X systems. The pre-installed version may not be the most recent one v3.12.3 (31/07/2024) and the latest version can be downloaded from <http://python.org/download/>.

Upgrade the Package Installer for Python (**pip**).

```
~$ sudo apt upgrade python3-pip
```

Check outdated Python packages using **pip**.

```
~$ pip list --outdated
```

Package	Version	Latest	Type
attrs	23.2.0	24.2.0	wheel
Babel	2.10.3	2.16.0	wheel
blinker	1.7.0	1.8.2	wheel
certifi	2023.11.17	2024.8.30	wheel

Confirm the versions of **python3** and **pip**.

```
~$ python3 --version
Python 3.12.3
```

```
~$ python3 -m pip --version
pip 24.0 from /usr/lib/python3/dist-packages/pip (python 3.12)
```

5.2 Python on Microsoft Windows

Unlike GNU/Linux, Python does not come pre-installed on Microsoft Windows by Default. Python for Windows can be downloaded from:

<https://python.org/downloads/windows/>

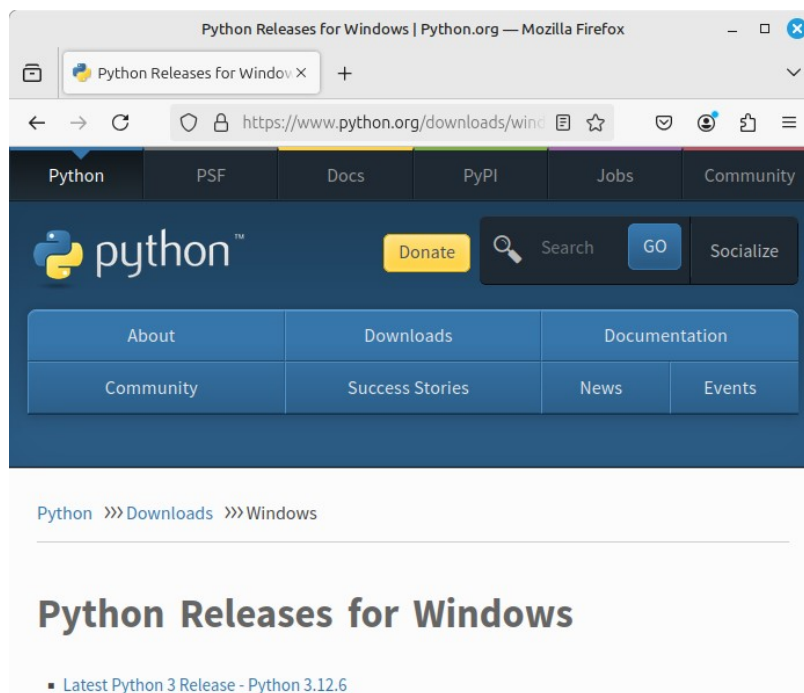


Illustration 6: Python for Microsoft Windows

Download the **Windows Installer (64-bit)** and install it. Make sure to select the option to **Add Python 3.12 to PATH**.

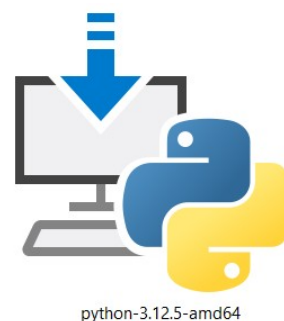
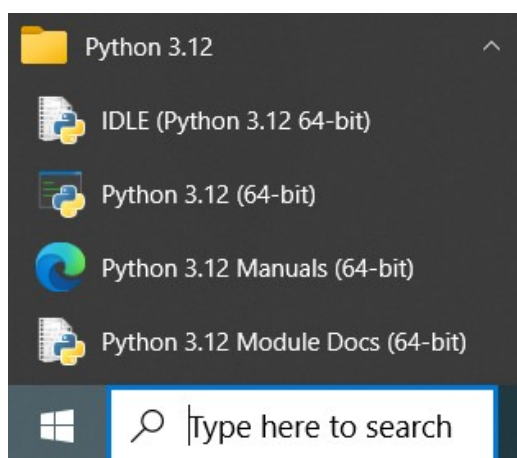


Illustration 7: Python installed on Microsoft Windows

6. Running Python

6.1 The Python Interpreter on GNU/Linux - python3

Typical Python implementations offer both an interpreter and compiler. There is an interactive interface to python with a **Read - Evaluate - Print Loop (REPL)** loop. Here the current version python3 is demonstrated.

```
~$ python3
Python 3.12.3 (main, Jul 31 2024, 17:43:48) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> xlist = list()
>>> for x in range(1,5):
...     xlist.append(x*x)
...
>>> print(xlist)
[1, 4, 9, 16]
>>>
>>> [pow(x,2) for in range(1,5)]
[1, 4, 9, 16]
>>>
```

6.2 The Python Interpreters on Microsoft Windows, python

On Microsoft Windows the Python installation is also Python3; however, the interpreter is simply called **python**.

```
C:\Users\LOVELACE\Desktop> python --version
Python 3.12.3

C:\Users\LOVELACE\Desktop> python -m pip --version
pip 22.2.1 from C:\Users\LOVELACE\AppData\Local\Programs\Python\Python310\
lib\site-packages\pip (python 3.12)

C:\Users\LOVELACE\Desktop> python
Python 3.12.3 (main, Jul 31 2024, 17:43:48) [MSC v.1932 64 bit (AMD64)] on
win32
Type "help", "copyright", "credits" or "license" for more information.

>>> xlist = list()
>>> for x in range(1, 5):
...     xlist.append(x*x)
...

>>> print(xlist)
[1, 4, 9, 16]

>>> [x*x for x in range(1, 5)]
[1, 4, 9, 16]
>>>
```

6.3 iPython

iPython can be used as a replacement for the standard Python shell, or it can be used as a complete working environment for scientific computing (like Matlab or Mathematica) when paired with the standard Python scientific and numerical tools. It supports dynamic object introspections, numbered input/output prompts, a macro system, session logging, session restoring, complete system shell access, verbose and coloured traceback reports, auto-parentheses, auto-quoting, and is embeddable in other Python programs.

Install the iPython enhanced interactive Python3 shell.

```
~$ sudo apt install ipython3
```

Running the iPython enhanced interactive Python3 shell.

```
~$ ipython3
Python 3.12.3 (main, Jul 31 2024, 17:43:48) [GCC 13.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.20.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: for x in range(5):
...:     print('iPython rocks')
...:
iPython rocks
iPython rocks
iPython rocks
iPython rocks
iPython rocks

In [2]: quit()
```

6.4 bPython

bPython is another replacement for the standard Python shell, it supports: In-line syntax highlighting, Readline-like autocomplete with suggestions displayed as you type, Expected parameter list for any Python function, "Rewind" function to pop the last line of code from memory and re-evaluate, Send the code entered to a pastebin, save the code entered to a file and auto-indentation.

Install the bPython enhanced interactive Python3 shell.

```
~$ sudo apt install bpython
```

Running the bPython enhanced interactive Python3 shell.

```
~$ bpython
bpython version 0.24 on top of Python 3.12.3 /usr/bin/python3
>>> for x in range(5):
...     print('bPython rocks even more')
...
...
bPython rocks more
bPython rocks more
bPython rocks more
bPython rocks more
bPython rocks more
>>> quit()
```

7. Integrated Development and Learning Environment (IDLE)

Python comes with a large library of standard modules and there are several options for an Integrated Development Environment (IDE). Integrated Development and Learning Environment (IDLE) is an integrated development environment for Python, which has been bundled with the default implementation of the language.

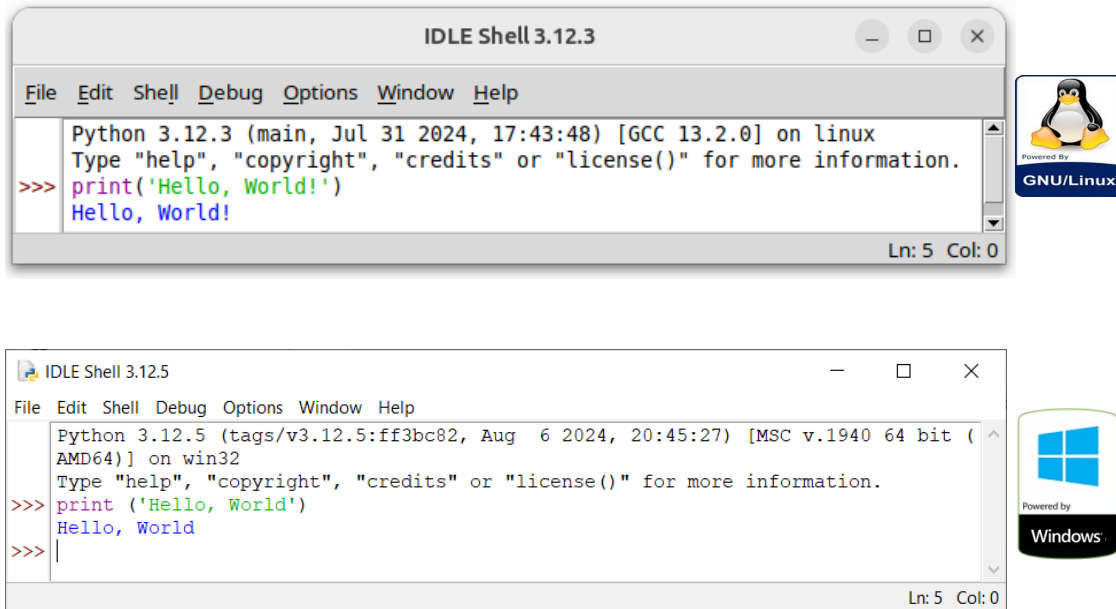


Illustration 8: IDLE

IDLE offers a python shell with syntax highlighting, an integrated debugger with stepping, persistent breakpoints, and call stack visibility. It works well with Microsoft Windows. On GNU/Linux and UNIX most editors, **gedit**, **emacs**, etc... can interpret Python. **Eclipse** with **Pydev** (<http://pydev.sourceforge.net/>) is also an IDE option.

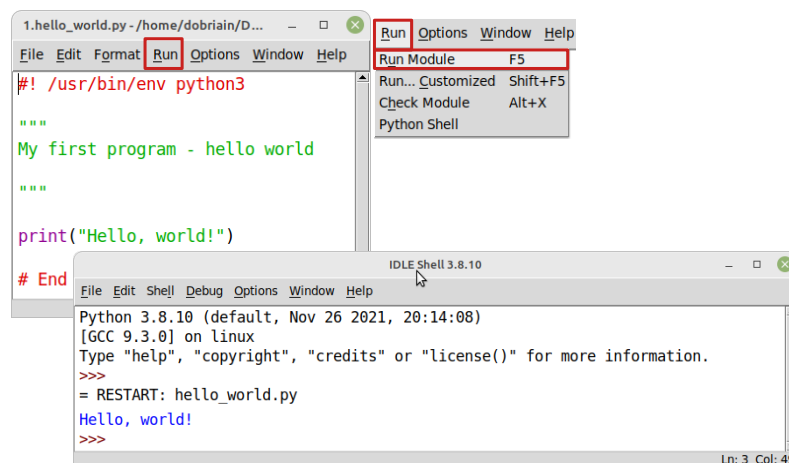
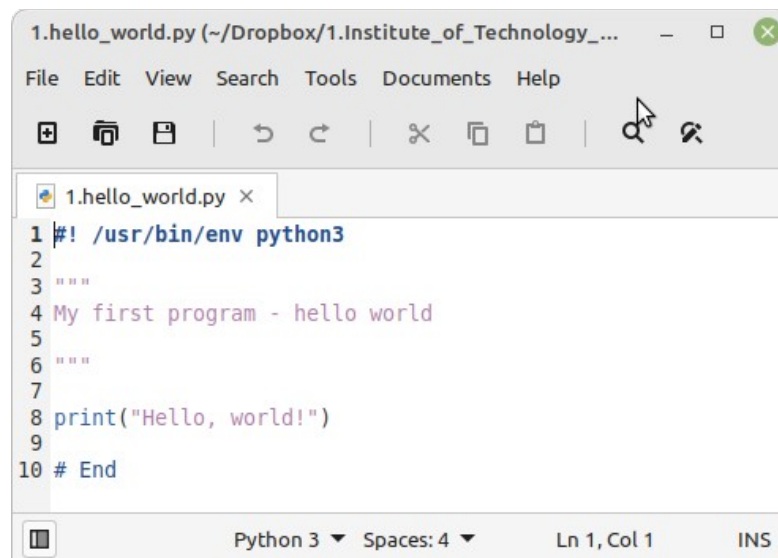


Illustration 9: IDLE Editor and Run module option

Python programs can also be edited by a variety of editors. Here is an example of editing Python with **gedit**.



```
1.hello_world.py (~/.Dropbox/1.Institute_of_Technology_...  -  □  ×)
File Edit View Search Tools Documents Help
+  [Icons]  [Icons]  [Icons]  [Icons]  [Icons]  [Icons]  [Icons]  [Icons]
1.hello_world.py ×
1 #!/usr/bin/env python3
2
3 """
4 My first program - hello world
5
6 """
7
8 print("Hello, world!")
9
10 # End
Python 3 Spaces: 4 Ln 1, Col 1 INS
```

Illustration 10: hello_world.py in gedit

```
~$ chmod +x hello_world.py
~$ ./hello_world.py
Hello world
```

Running interactively on GNU/Linux and UNIX.

```
~$ python3
>>> 3 + 3
6
```

In interactive mode python prompts with '>>>'. To exit Python (not Idle) in GNU/Linux or UNIX, type **CONTROL-D**, while in Microsoft Windows, type **CONTROL-Z + <Enter>** is required. An alternative is to type **exit()**.

```
>>> exit()
~$
```

7.1 Text editors and Integrated Development Environment (IDE)

Python programs can be edited with IDLE or a simple text editor. However, it makes it much easier if the editing tool has syntax highlighting, such an example is **gedit**, the in-build text editor in the Gnome Desktop Environment, typically Linux only. This text editor performs syntax highlighting for python2 and python3 and is easy to use.

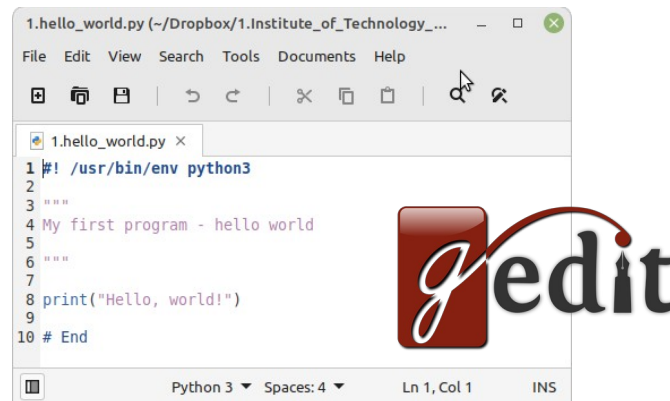


Illustration 11: Gnome Gedit

An IDE on the other hand is a software application that provides comprehensive facilities to programmers for software development. An IDE normally consists of at least a source code editor, build automation tools and a debugger.

Two options to explore further are **CudaText**, and **Sublime text**.

7.1.1 CudaText

CudaText is an Open source project and free of cost. It is a cross-platform native GUI text and source code editor. The program is extensible by Python add-ons (plugins, linters, code tree parsers, external tools). It can be downloaded from:

<http://uvviewsoft.com/cudatext/>

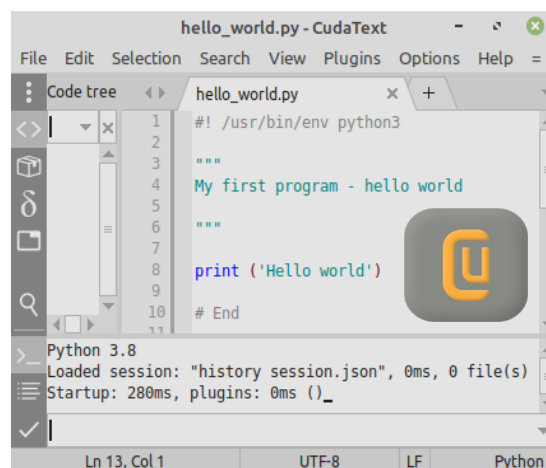


Illustration 12: CudaText

7.1.2 Sublime text

Sublime text is a favourite among programmers for many years. it is a shareware cross-platform source code editor with a Python application programming interface. It can be tried for free however ongoing use requires a license which costs \$80 USD. For GNU/Linux it is located in the distribution repositories. For other platforms download and install from: <https://www.sublimetext.com>

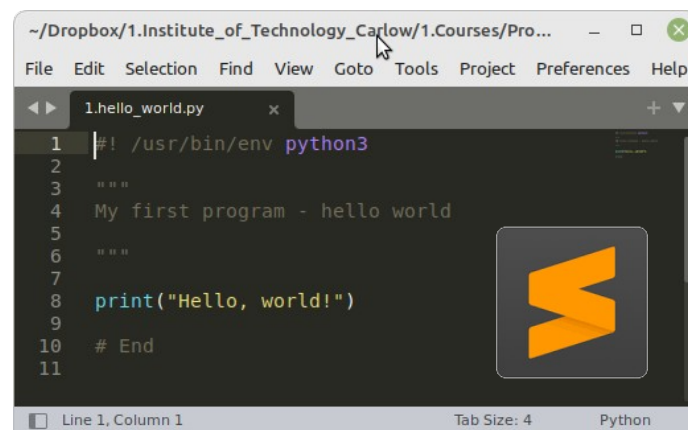


Illustration 13: Sublime text

On Linux install:

```
~$ sudo apt-get install -y sublime-text
```

7.1.3 Notepad++ on Microsoft Windows

Notepad++ is a free open source GNU editor for Microsoft Windows that offers features similar to editors on GNU/Linux.

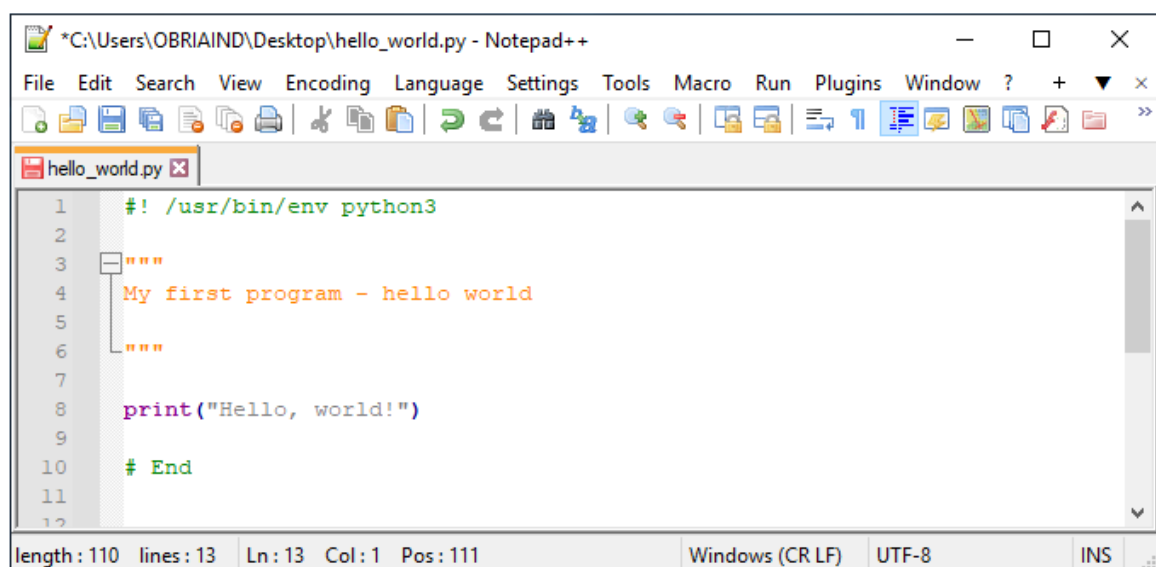


Illustration 14: Notepad++

8. Running Programs

On GNU/Linux and UNIX call the python program via the python interpreter as follows:

```
~$ python my_program.py
```

Make a python file directly executable by adding the appropriate path to the python interpreter as the first line of the file (**#!** shebang line). The use of **env** as an executable in **/usr/bin**, is constant in all GNU/Linux and UNIX distributions. Using **#!/usr/bin/env python** instead of the absolute path **#!/usr/bin/python** ensures that python is found, in case it might not be in exactly the same location across different GNU/Linux or UNIX distributions.

```
#!/usr/bin/env python3
or
```

```
#!/usr/bin/python3
```

The next step is to make the file executable.

```
~$ chmod a+x my_program.py
```

It is not possible to invoke the Python program file from GNU/Linux or UNIX command line without calling **python3** or in Microsoft Windows shell, **python**.

```
~$ ./my_program.py
```

Take this example script: **factorial.py**. Here is the flowchart. Note the function is shown, almost as another program. The function $f(x)$ is both from within the main program and from within the function itself.

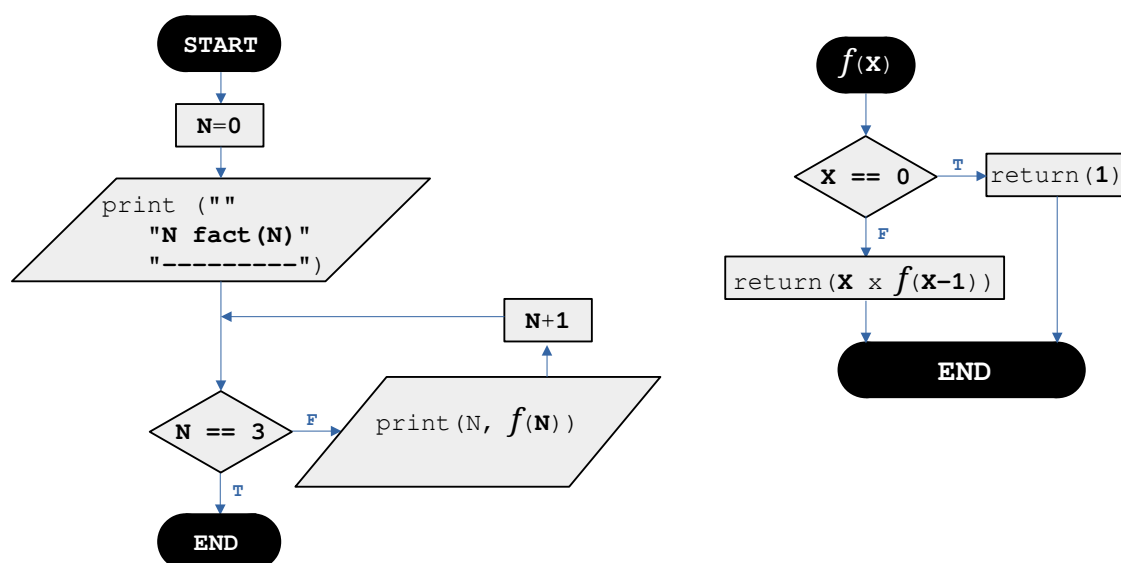


Illustration 15: factorial.py flowchart

Consider the execution, how was the output achieved.

```

1  #!/usr/bin/env python3
2
3
4  def fact(x):
5      # Returns the factorial of its argument
6      if x == 0:
7          return 1
8      return x * fact(x - 1)
9
10
11 print("")
12 print("N fact(N)")
13 print("-----")
14
15 for n in range(10):
16     print(n, fact(n))
17
18 # End

```

```

~$ chmod a+x factorial.py
~$ ./factorial.py

N fact(N)
-----
0 1
1 1
2 2
3 6
4 24
5 120
6 720
7 5040
8 40320
9 362880

```

Illustration 16: `factorial.py` ran on GNU/Linux

Consider Illustration 17, in the case of the first loop in the main program the `fact(N)` function is called with `N = 0`, therefore the function returns a `1` as the `if x == 0` is true. The main program prints `0 1`

However, when `N = 1`, `fact(N)` tries to return `1 + fact(1-1)`, it cannot so it calls another instance of `fact(N)` with `N = 0`, which returns a `1`. Now the first instance of the `fact(N)` function returns `1x1` or simply `1`. The main program prints `1 1`.

In the third case when `N = 2`, there is a requirement for 3 instances of the `fact(N)` function, one called from the main program, one from the first `fact(N)` function instance and another from the second `fact(N)` function instance. The third `fact(N)` function instance returns `1`. The second `fact(N)` function instance multiplies the `1` received with the `1` it has and returns `1` to the first instance. This inner instance multiplies `1` to the `2` it has and returns `2` to the main program which prints `2 2`.

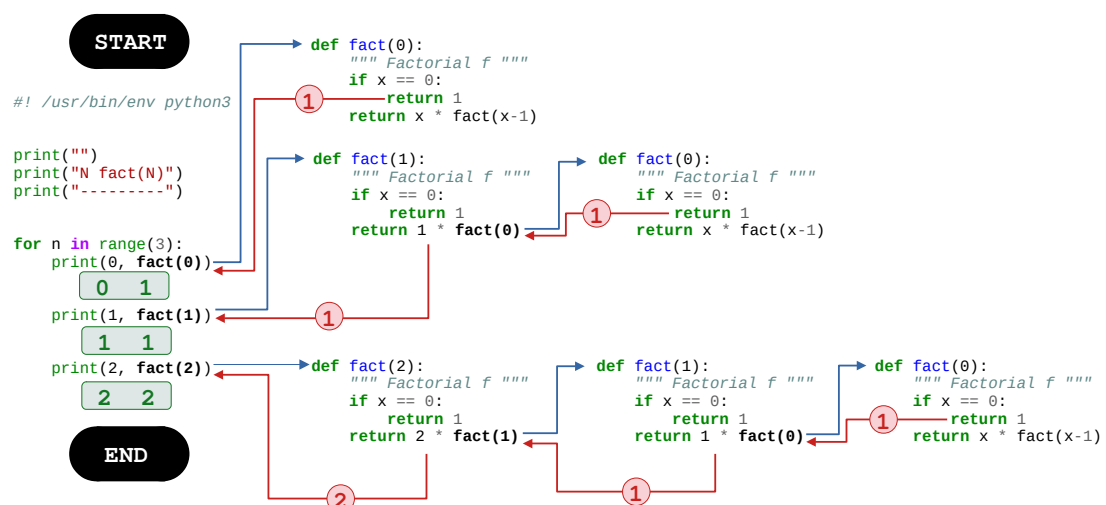
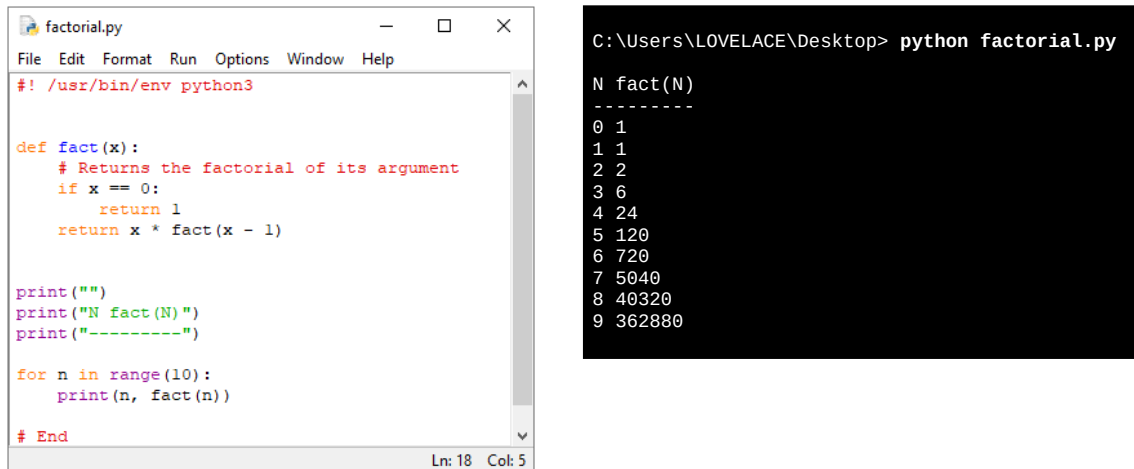


Illustration 17: `factorial.py` loops through function

Here the **factorial.py** program is ran on Microsoft Windows.



The image shows a Windows environment. On the left, a text editor window titled 'factorial.py' displays the following Python code:

```
#!/usr/bin/env python3

def fact(x):
    # Returns the factorial of its argument
    if x == 0:
        return 1
    return x * fact(x - 1)

print("")
print("N fact(N)")
print("-----")

for n in range(10):
    print(n, fact(n))

# End
```

On the right, a black command prompt window shows the command `python factorial.py` being executed, resulting in the following output:

```
C:\Users\LOVELACE\Desktop> python factorial.py

N fact(N)
-----
0 1
1 1
2 2
3 6
4 24
5 120
6 720
7 5040
8 40320
9 362880
```

Illustration 18: factorial.py ran on Microsoft Windows

When a Python program is called from the shell or command line, the interpreter evaluates each expression in the file. Familiar mechanisms are used to provide command line arguments and/or redirect input and output. Python also has mechanisms to allow a program to act both as a script and as a module to be imported and used by another python program.

Consider this example script that reads in emails and extracts email addresses from the file using regular expressions. While the detail of the regular expression will not make sense as yet it will be covered in a later section of the course.

```
~$ cat RE-script.py
1  #!/usr/bin/env python3
2
3  """
4  Reads text from standard input and outputs any email
5  addresses it finds, one to a line.
6  """
7
8  import re
9  from sys import stdin
10
11 # a regular expression ~ for a valid email address
12 pat = re.compile(r"([a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+)")
13
14 for line in stdin.readlines():
15     for address in pat.findall(line):
16         print(address)
17
18 # // End //
```

Illustration 19: RE-script.py

```
~$ ./RE-script.py < email.txt
ame-bounces@tus.ie
tom.ryan@tus.ie
EEINC@tus.ie
electronics@tus.ie
michelle.cleare@tus.ie
tom.ryan@tus.ie
```

```
~$ cat unique-sorted.py
1  #!/usr/bin/env python3
2
3  import re
4  from sys import stdin
5
6  # a regular expression ~ for a valid email address
7  pat = re.compile(r"([a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-\.]+)")
8  # found is an initially empty set (a list w/o duplicates)
9  found = set()
10 for line in stdin.readlines():
11     for address in pat.findall(line):
12         found.add(address)
13 # sorted() takes a sequence, returns a sorted list of its elements
14 for address in sorted(found):
15     print(address)
16
17 # // End //
```

Illustration 20: unique_sorted_list.py

Further enhancing the script to return a unique, sorted list as shown. Line 9 creates an empty set called **found**. A set is an unordered collection of items. Each element is unique. The set is mutable in that items can be added or removed from the set. Line 10 is a loop that reads in lines from the **email.txt** file and assigns each line to the variable **line**. An inner loop at line 11 checks the line for the presence of email addresses via the regular expression **pat** and if found line 12 adds the address to the set **found**. Line 14 loops through a list created by a sort of the set **found** and line 15 prints each email in turn.

```
~$ ./unique_sorted_list.py < email.txt
EEINC@tus.ie
ame-bounces@tus.ie
electronics@tus.ie
michelle.cleare@tus.ie
tom.ryan@tus.ie
```

Question: Why do you think the address **EEINC@tus.ie** is at the top of the list?

9. Mathematical Operators

Python has basic mathematical operators. It is simple to execute simple mathematical operations on the Python shell as follows. Note that there are two divide operators, one returns a number of type **float** even if there is no remainder and the other returns an **integer**.

```
>>> 5+3          # Add
8

>>> 5-3          # Subtract
2

>>> 5*3          # Multiply
15

>>> 5**3         # Exponent
125

>>> 27/4         # Divide, returns a float
6.75

>>> 27//4        # Divide, returns integer
6

>>> 27%4         # Modulo, returns the remainder
3
```

It is even possible to run a single line command without calling the inline interpreter.

```
~$ python3 -c 'print(3+5)'
8
```

Assigning mathematical results to variables

```
>>> a = 5+3
>>> b = a-3
>>> c = b%2

>>> print(a, b, c)
8 5 1

>>> d = a/2
>>> e = a//2

>>> print(type(d),d)
<class 'float'> 4.0
>>> print(type(e),e)
<class 'int'> 4
```


10. Simple functions

```

~$ cat exmod.py
1 """factorial done recursively and iteratively"""
2
3
4 def fact1(n):
5     """First factorial function"""
6     ans = 1
7     for i in range(2, n):
8         ans = ans * n
9     return ans
10
11
12 def fact2(n):
13     """Second factorial function"""
14     if n < 1:
15         return 1
16     else:
17         return n * fact2(n - 1)
18
19
20 # // End //

~$ cat execute-exmod.py
1 #! /usr/bin/env python3
2
3 import exmod
4
5 print(exmod.fact1(6))
6 print("\n")
7 print(exmod.fact2(200))
8 print("\n")
9 print(exmod.fact1)
10
11 # // End //

```

Illustration 21: Simple functions

It is possible for a python script to import code from another python script. This is very common with functions. Here **exmod.py** is called by **execute-exmod.py** which imports the code of **exmod.py** into the **execute-exmod.py** code. Running the **execute-exmod.py** program demonstrates that it in fact used the functions from within **exmod.py**.

```

~$ ./execute-exmod.py
1296

```

```

7886578673647905035523632139321850622951359776871732632947425332443
5944996340334292030428401198462390417721213891963883025764279024263
7105061926624952829931113462857270763317237396988943922445621451664
2402540332918641312274282948532775242424075739032403212574055795686
60226031904170324062351700858796178922222789623703897374720000000000
0000000000000000000000000000000000000000000000000000000000000000

```

```

<function fact1 at 0x7fca6dc048c8>

```

Note that python automatically compiles a script to compiled byte code, before running it. When a module is imported for the first time a bytecode, **.pyc**, file, in a directory **__pycache__** containing the compiled code will appear in the same directory as the **.py** file. Note the **exmod.cpython-312.pyc** file created in the **__pycache__** directory.

Function files that are accessible by python programs should be placed in the python path.

```

>>> import sys
>>> sys.path
['', '/usr/lib/python312.zip', '/usr/lib/python3.12', '/usr/lib/python3.12/lib-
dynload', '/home/ubuntu/.local/lib/python3.12/site-packages',
'/usr/local/lib/python3.8/dist-packages', '/usr/lib/python3/dist-packages']

```

11. Getting help on functions within a module

Help can easily be obtained for functions within a module.

```
>>> import exmod
>>> help(exmod)
Help on module exmod:

NAME
    exmod - factorial done recursively and iteratively

FUNCTIONS
    fact1(n)
        First factorial function

    fact2(n)
        Second factorial function

FILE
    /usr/lib/python3/dist-packages/exmod.py
```

12. Bytecode files

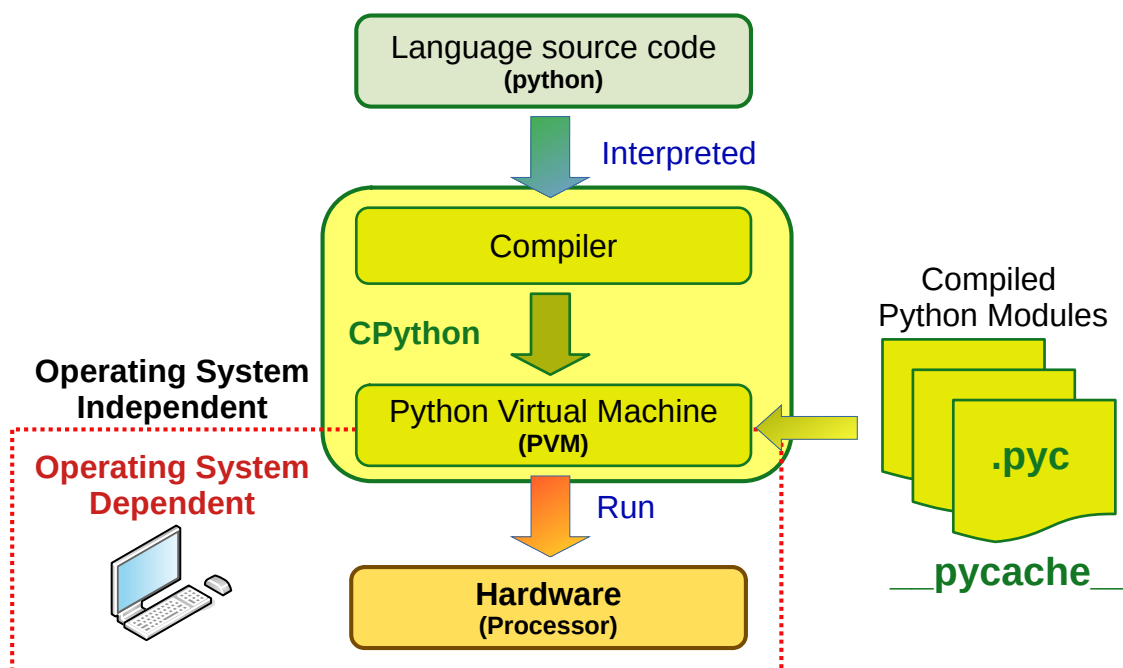


Illustration 22: Python Virtual Machine

At runtime python compiles the **.py** files and saves the result as **.pyc** files, so it can reference them in subsequent invocations. The **.pyc** file contains compiled bytecode of python source files output from the python interpreter. This code is then executed by Python's Virtual Machine (PVM).

While python is an interpreted language, as opposed to a compiled one such as C/C++, the distinction is blurred due to the presence of the bytecode compiler. Compiling means converting a program to machine code. Interpreters, however, take the text form of the program and execute it statement by statement.

For example, When the **hello_world.py** source file is ran, the python interpreter first looks to see if any **hello_world.pyc** exists, and if it is more recent than **hello_world.py**. If so, the interpreter runs it. If it not, or **hello_world.py** is more recent than it, the interpreter first compiles **hello_world.py** to **hello_world.pyc** before execution.

13. Exercise #1.2

1. Install **python3** and **pip3**.
2. Review help for the **print()** function.
3. Print the **sys.path** to the shell.
4. Write a small program to print your name to the shell.
5. Make the program executable.
6. Run the program.

This page is intentionally blank