

Beginners to Advanced

A stylized illustration of a person with dark hair, seen from the side, sitting at a desk and working on a laptop. The person is wearing a light blue long-sleeved shirt. The laptop screen displays a document with text and a small Python logo. The desk is dark blue and features a potted plant with green leaves and a white mug. The entire scene is set within a circular frame. Surrounding this central circle are numerous colorful icons representing various concepts: a lightbulb (idea), a cloud with a circular arrow (cloud services), a gear inside a head (thought), a magnifying glass (search), a circular arrow (refresh/cycle), a microchip (technology), a database cylinder (data), a key (security), a USB symbol, a folder, a document with a checkmark, a right-angle triangle, a square, a circle, a zigzag line, a square with a cross, a square with an arrow, a square with a plus, a square with a minus, a square with a multiply, a square with a divide, a square with a percent, a square with a hash, a square with a dollar sign, a square with a pound sign, a square with a yen sign, a square with a euro sign, a square with a ruble sign, a square with a won sign, a square with a new sheqel sign, a square with a new dollar sign, a square with a new peso sign, a square with a new real sign, a square with a new zloty sign, a square with a new forint sign, a square with a new koruna sign, a square with a new lek sign, a square with a new dracma sign, a square with a new tenge sign, a square with a new dirham sign, a square with a new sheqel sign, a square with a new dollar sign, a square with a new peso sign, a square with a new real sign, a square with a new zloty sign, a square with a new forint sign, a square with a new koruna sign, a square with a new lek sign, a square with a new dracma sign, a square with a new tenge sign, a square with a new dirham sign.

Course Contents



- Setup Development Tools
- Introduction to Python
- Syntax, Variables And Keywords
- Input And Output : Usage Of End Parameter & Formatted Printing
- Statements, Indentation And Comments In Python
- Operators
- Data Types And Type Casting
- Strings & its methods
- Decision Making
- Conditional Branching
- Loops, Range Function, Membership Operators
- Break, Continue and Return statements
- Switch Case
- Main Method
- Functions
- Global Vs Local Variables
- Lists, Tuples, Sets, Dictionary and Arrays
- Copy Methods
- Exception Handling
- OOPS : Class, Objects etc
- File Handling
- Modules: OS, JSON etc



Installing Tools





Code Editors and IDEs

Code Editors

- Code editors/text editors are great for writing and editing the code
- They are usually lightweight and suitable for development of small projects
- However, once your program gets larger, you need much more capabilities in the editors, like test and debug your code, other than editing. That's where IDEs come into the play
- Depending upon the language one codes on the editor, it highlights special keywords and gives some suggestions

Ex: Visual Studio Code, Sublime Text, Atom, Vi, Vim etc

IDE (Integrated Development Environment)

- An IDE is a software that consists of common developer tools into a single user-friendly GUI (Graphical User interface)
- An IDE majorly consists of a source code editor for writing software code, local build automation for creating a local build of the software like compiling source code
- A good IDE must possess: Save and Reload Source Code , Execution from Within the Environment, Debugging Support, Syntax Highlighting, Automatic Code Formatting, Auto code Completion etc
- Some IDEs also include source control and support for package managers like PIP

Ex: IDLE, PyCharm, Jupyter, Spyder are some Python IDEs



Python interpreter should be installed before using any of the above editors and IDEs



/kunchalavikram1427



Code Editors and IDEs

Installing IDLE

- IDLE (Integrated Development and Learning Environment) is a default editor that comes with Python
 - It is one of the best Python IDE software which helps a beginner to learn Python easily
 - It is optional for many Linux distributions and can be installed by package managers like **APT**, **YUM** etc
 - It consists of a multi-window text editor with syntax highlighting and integrated debugger with stepping, persistent breakpoints, and call stack visibility
-
- Open a browser window and navigate to <https://www.python.org/downloads/> to download the latest python installer

The image shows two overlapping windows. The background window is titled 'Python 3.5.1 Shell' and displays the Python interpreter's startup message: 'Python 3.5.1 (default, Jul 10 2016, 20:36:01) [GCC 6.1.1 20160621 (Red Hat 6.1.1-3)] on linux'. It prompts the user to type 'copyright', 'credits', or 'license()' for more information. The foreground window is titled 'hello.py - /home/user/hello.py (3.5.1)' and shows a Python script with the following content:

```
#!/usr/bin/python3  
  
print("Hello Fedora 24")
```



Code Editors and IDEs

Installing IDLE

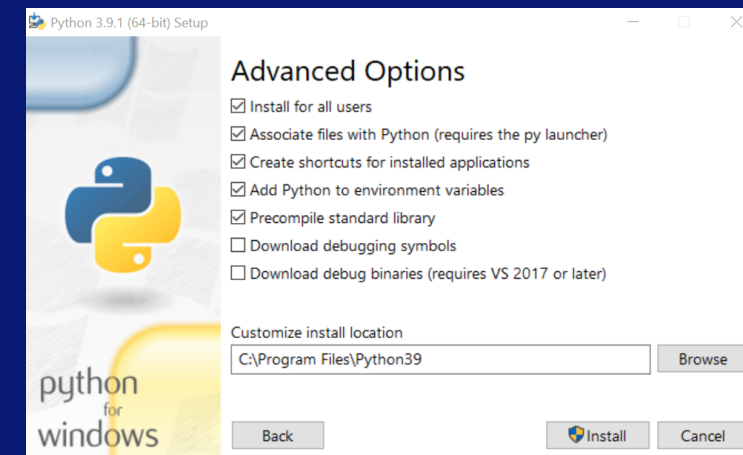
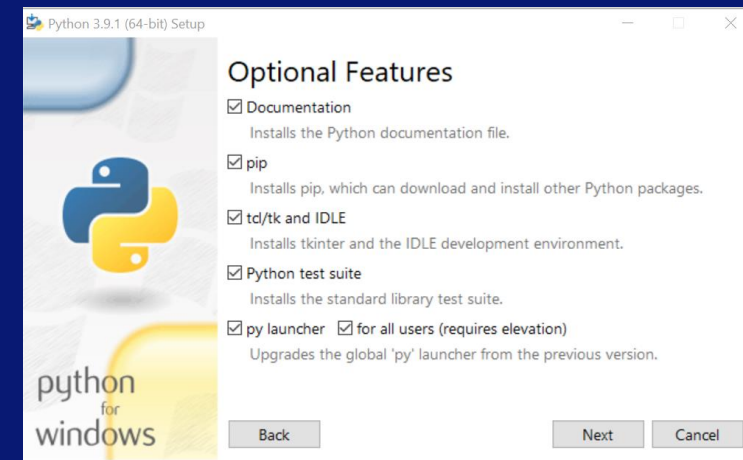
For Windows

- Open a browser window and navigate to <https://www.python.org/downloads/> to download the latest python installer
- Double click on the downloaded file and click on Run/Install
- Select the checkboxes as shown in the images
- Click on 'Customize installation' button to customize the installation location and which additional features get installed, including pip and IDLE
- The Install launcher for all users (recommended) checkbox is checked default. This means every user on the machine will have access to the py.exe launcher.
- Click on 'Install' to install the software
- To check the installation, run `python --version` or `py -3 --version` in the windows command prompt, which should display the version of python installed

```
Command Prompt
Microsoft Windows [Version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\428991>python --version
Python 3.9.1

C:\Users\428991>
```



 For Linux, follow the steps at <https://realpython.com/installing-python/>



Code Editors and IDEs

Visual Studio Code

- Visual Studio Code (aka VS Code) is a full-featured code editor available for Linux, Mac OS X, and Windows platforms
- Small and light-weight, but full-featured, VS Code is open-source, extensible, and configurable for almost any task
- Has support for more than 4700 extensions/plugins
- You can add a new language to the environment, such as **Python**, via a plugin
- Simply download and install the corresponding plugin to adapt it to the environment
- It has integrated powerful code auto-completion engine (IntelliSense), a debugging console, and a terminal to launch server commands
- VS Code will recognize your Python installation and libraries automatically



Using VS code for Python and setting up the editor, **worth reading!**
[https://code.visualstudio.com/docs/python/python-tutorial# prerequisites](https://code.visualstudio.com/docs/python/python-tutorial#prerequisites)

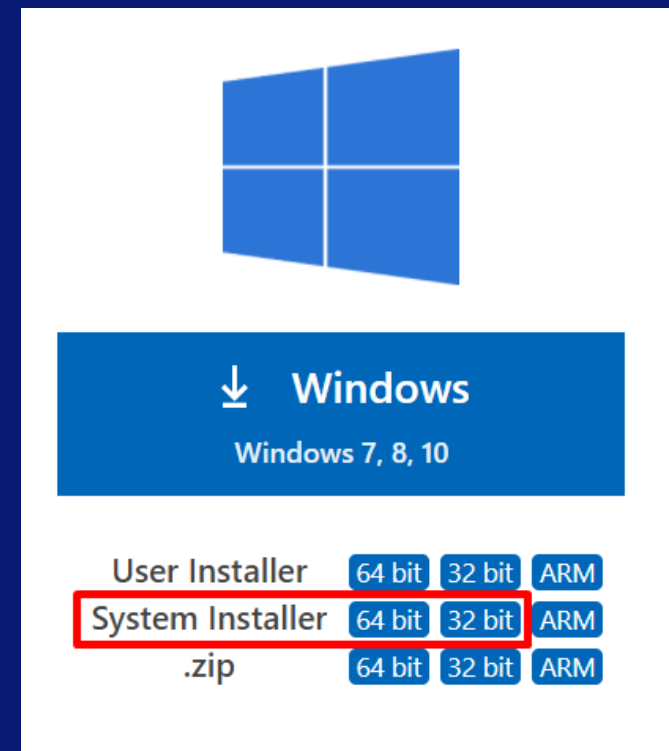
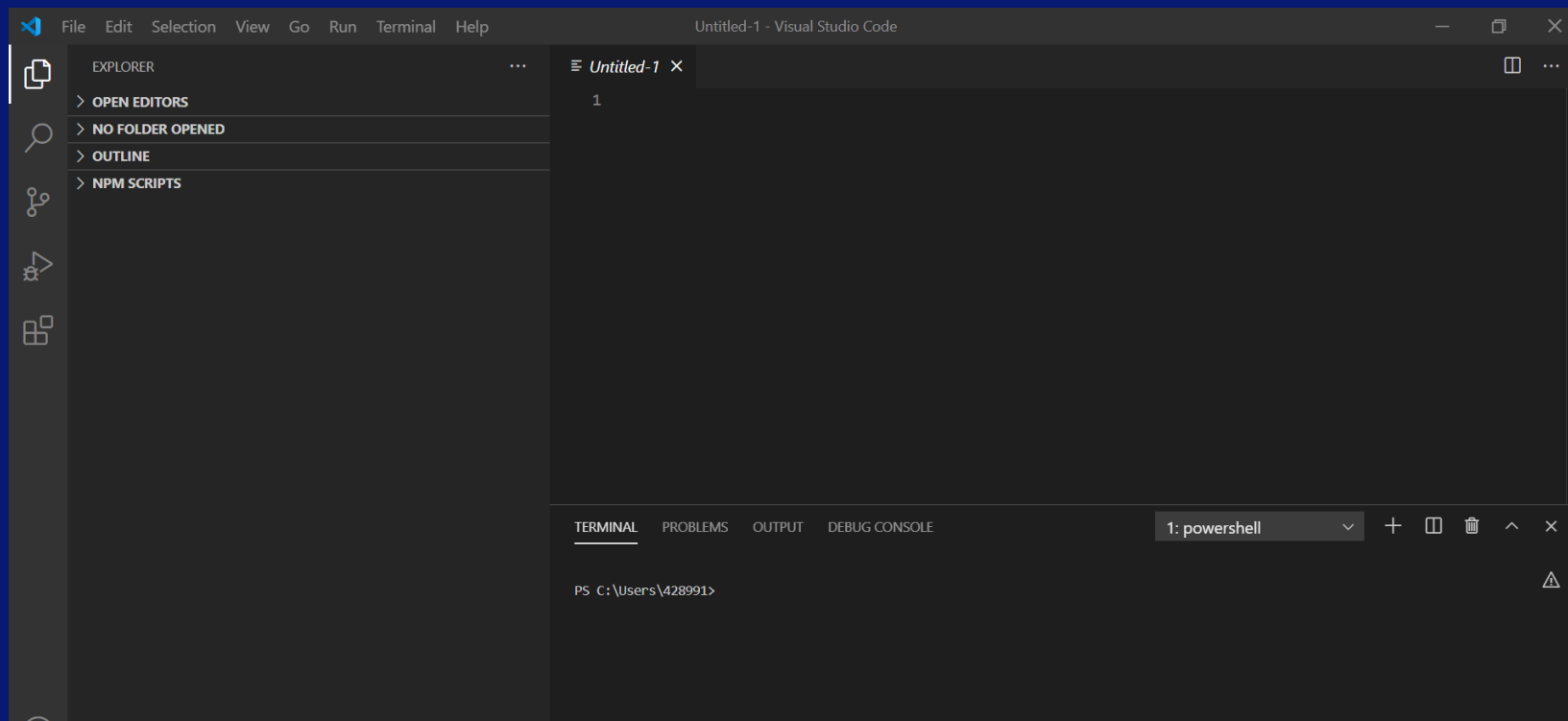




Code Editors and IDEs

Installing VS Code

- Go to <https://code.visualstudio.com/download> to download the latest version for your OS
- Download **System Installer 64/32 bit** depending on your OS arch
- Install with default settings and open the VS Code



Checkout these awesome tools for VS Code: <https://github.com/kunchalavikram1427/awesome-vscode>

Themes: <https://dev.to/thegeoffstevens/50-vs-code-themes-for-2020-45cc>



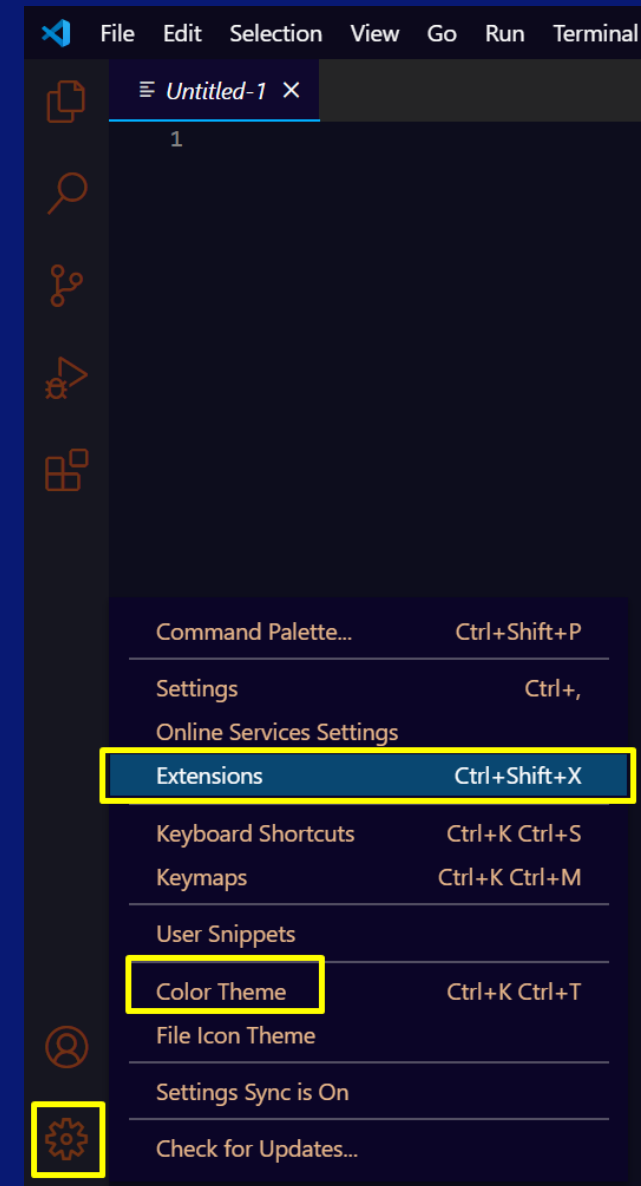
/kunchalavikram1427



Code Editors and IDEs

Installing VS Code Plugins

- VS Code comes with support for many plugins to extend its functionality for various languages
- For python, we need to install the following plugins from Extensions marketplace
 - Python
 - Pylance
 - Python Preview
 - Tabnine Autocomplete AI



Details on how to use the plugin are in the installation page of the plugin.
Checkout how to use [Python preview](#) plugin to debug the code as shown in installation page





Code Editors and IDEs

Installing VS Code Plugins

- **Python preview** plugin lets you debug the code and see the variables and other data directly in the IDE



More info on usage here

<https://marketplace.visualstudio.com/items?itemName=dongli.python-preview>

```
File Edit Selection View Go Run Terminal Help
helloworld.py X
c: > Users > 428991 > Desktop > helloworld.py > ...
1 list = [1,2,3,4,5,6]
2 reverse_list = []
3 for element in list:
4     reverse_list.append(element)
5 print(f"Reverse of original list is {reverse_list}")
6
```

Python 3.9.1.final.0

```
1 list = [1,2,3,4,5,6]
2 reverse_list = []
3 for element in list:
4     reverse_list.append(element)
5 print(f"Reverse of original list is {reverse_list}")
```

line that has just executed
next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

Print output (drag lower right corner to resize)

Frames

list
1

Objects

list
1

Global frame

list
1

reverse_list

reverse_list
1

element

element
1

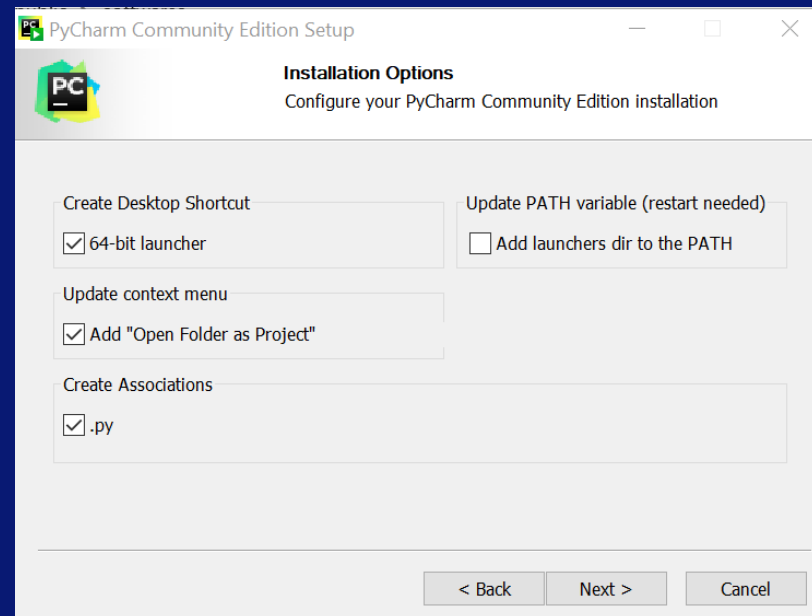
<< First < Back Step 5 of 16 Forward > Last >>



Code Editors and IDEs

PyCharm

- PyCharm is an integrated development environment developed by **JetBrains**
- Offers Community version(free) and the Professional version(paid) which offers advanced features such as full database management and a multitude of important Frameworks such as Django, Flask, Google App, Engine, Pyramid, and web2py
- The Community version offers different features such as syntax highlighting, auto-completion, debugging and live code verification
- Download PyCharm from <https://www.jetbrains.com/pycharm/download/> and install with below shown settings

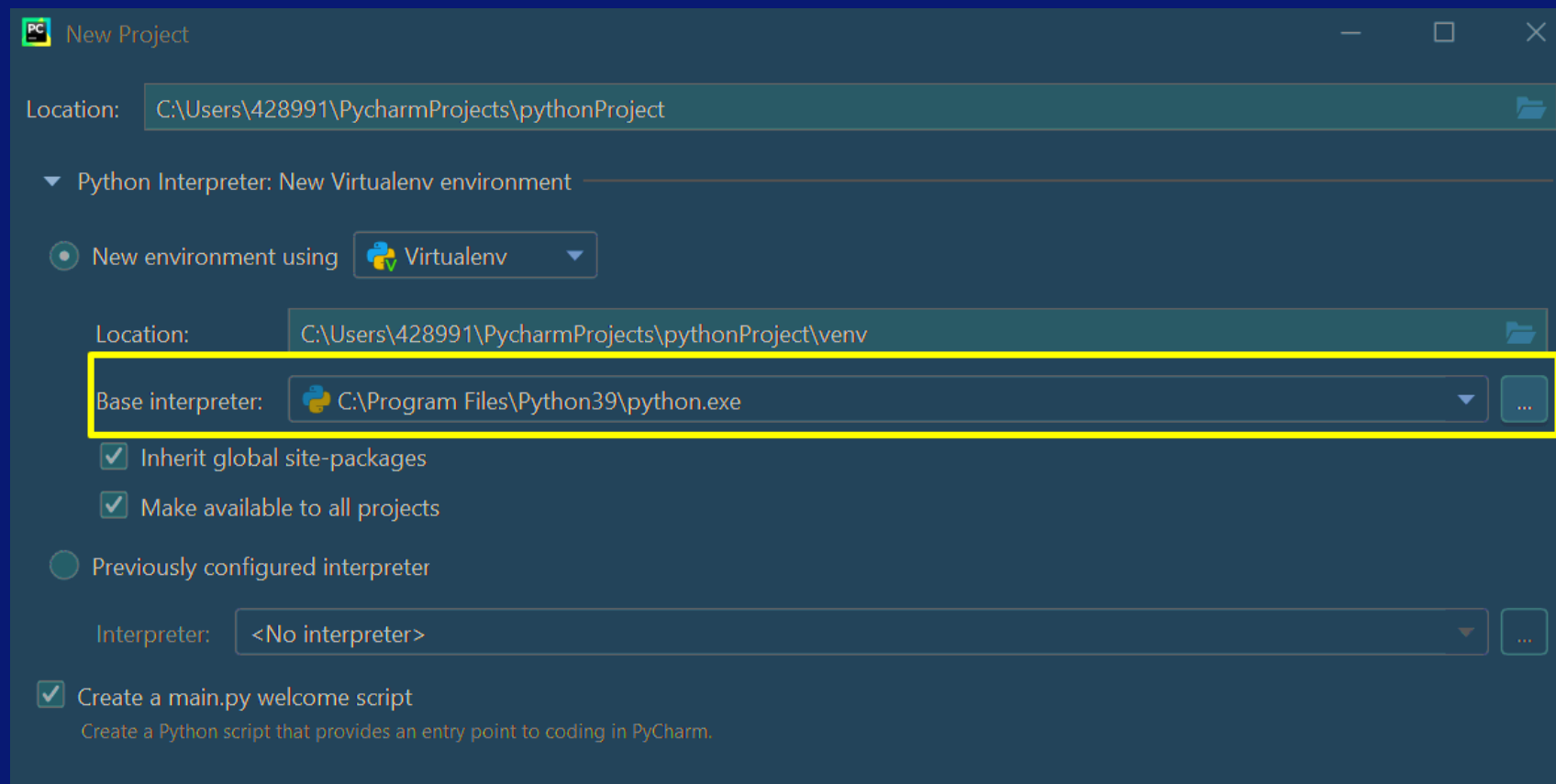
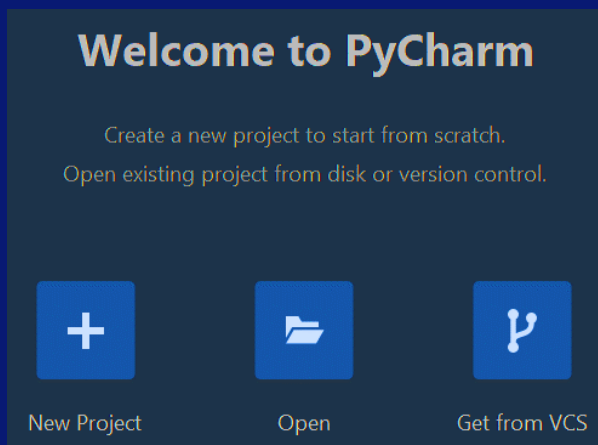




Code Editors and IDEs

PyCharm: Setting the Interpreter

- Open PyCharm and click on **New Project**
- Locate the base interpreter in the Python installation directory and click on **create**

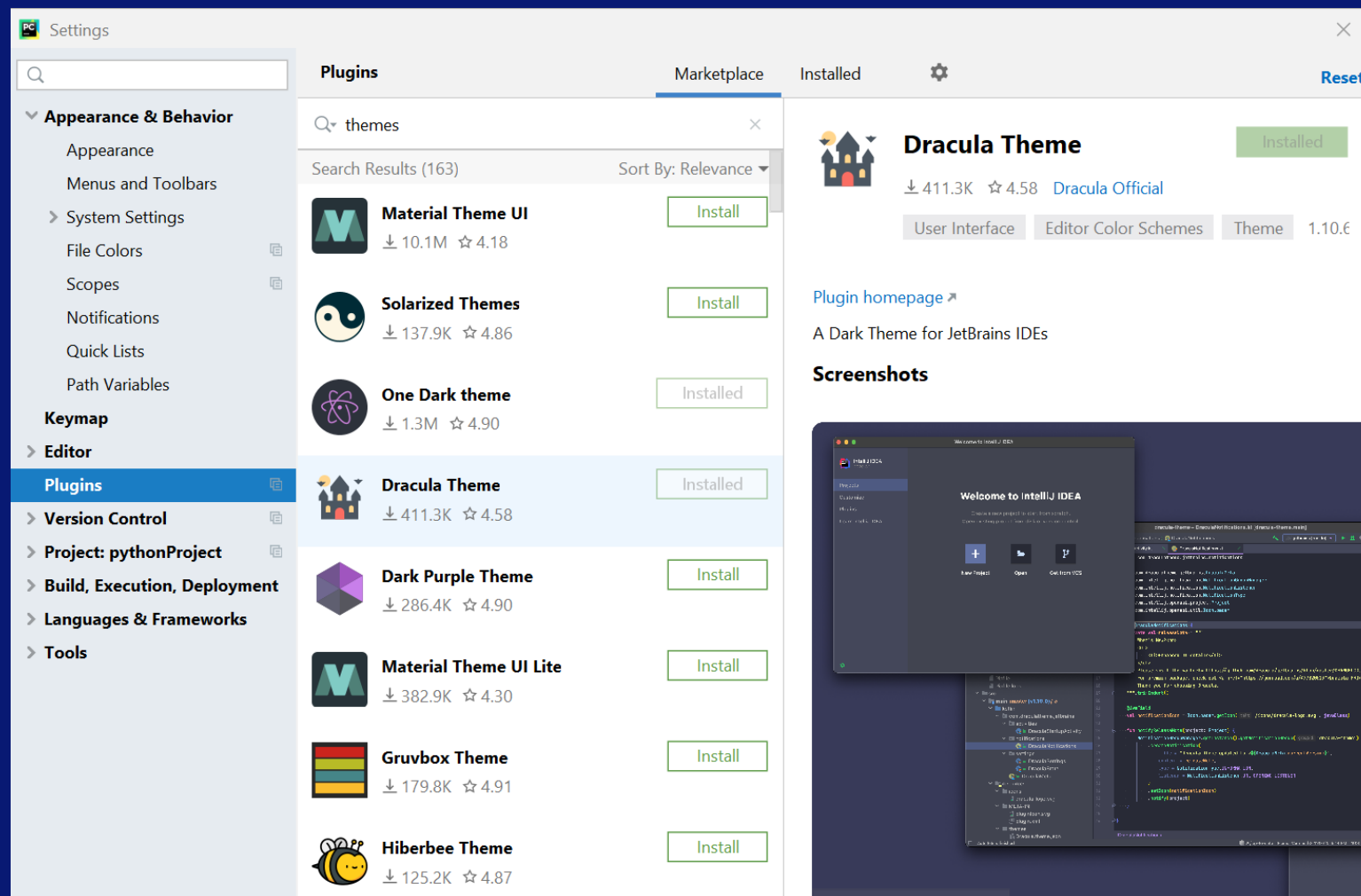




Code Editors and IDEs

PyCharm: Customize with plugins and themes

- Open PyCharm, click on **File** -> **Settings**





Code Editors and IDEs

Anaconda Distribution: Data science toolkit

- Anaconda is a distribution of the Python and R programming languages for scientific computing i.e., data science, machine learning applications, large-scale data processing, predictive analytics, etc
- It is the toolkit with thousands of open-source packages & libraries and simplifies package management and deployment
- Package versions in Anaconda are managed by the package management system **conda** and can be installed by **conda install** command
- Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source packages can be installed from **PyPI** using **PIP** as well as the **conda** package
- The difference between **conda** and the **PIP**(pip package manager) is in how package dependencies are managed
- When pip installs a package, it automatically installs any dependent Python packages without checking if these conflict with previously installed packages which results in erroneous results with existing code
- In contrast, conda analyses the current environment including everything currently installed, and shows a warning if this cannot be done



More info on **PyPI** and **PIP** in the later part of the course

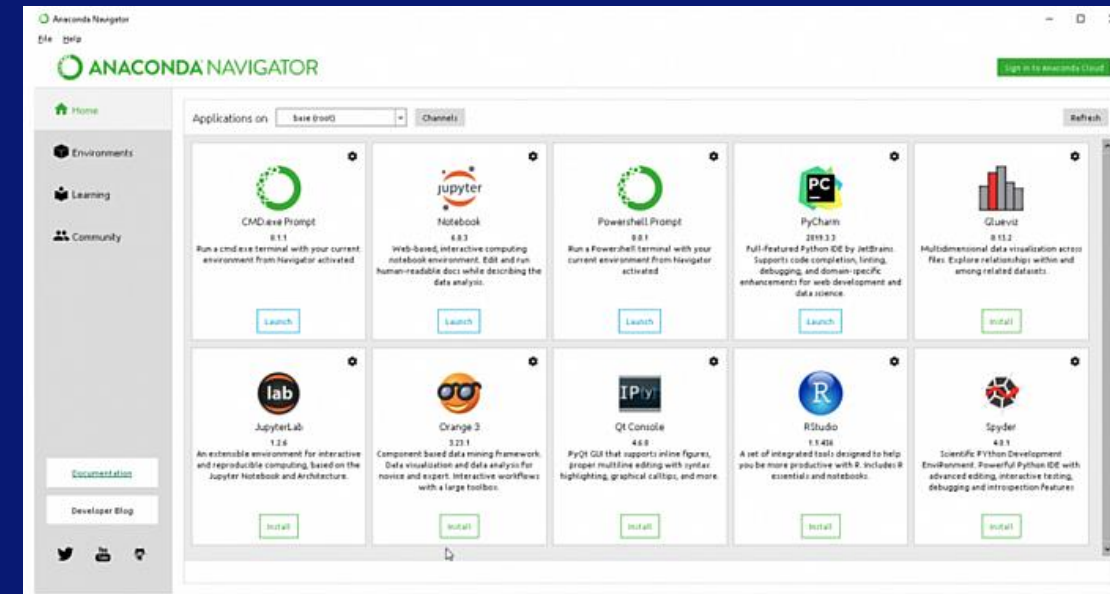




Code Editors and IDEs

Anaconda Navigator

- Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands
- Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them
- The following applications are available by default in the Navigator
 - Jupyter Lab
 - Jupyter Notebook
 - Qt Console
 - Spyder
 - PyCharm
 - Orange
 - RStudio
 - Visual Studio Code



Installing Anaconda

- Download from <https://www.anaconda.com/products/individual> and install it



Code Editors and IDEs

Jupyter Notebooks

- Jupyter notebook, formerly known as the **IPython notebook**, is a flexible tool that helps you create readable analyses, as you can keep code, images, comments, formulae and plots together for easy presentation
- It is one of the best Python IDE that supports for numerical simulation, data cleaning machine learning data visualization, statistical modelling and integrated data science libraries (matplotlib, NumPy, Pandas)
- Used for sharing documents that contain live code, equations, visualizations and narrative text
- Jupyter has support for over 40 different programming languages and Python is one of them

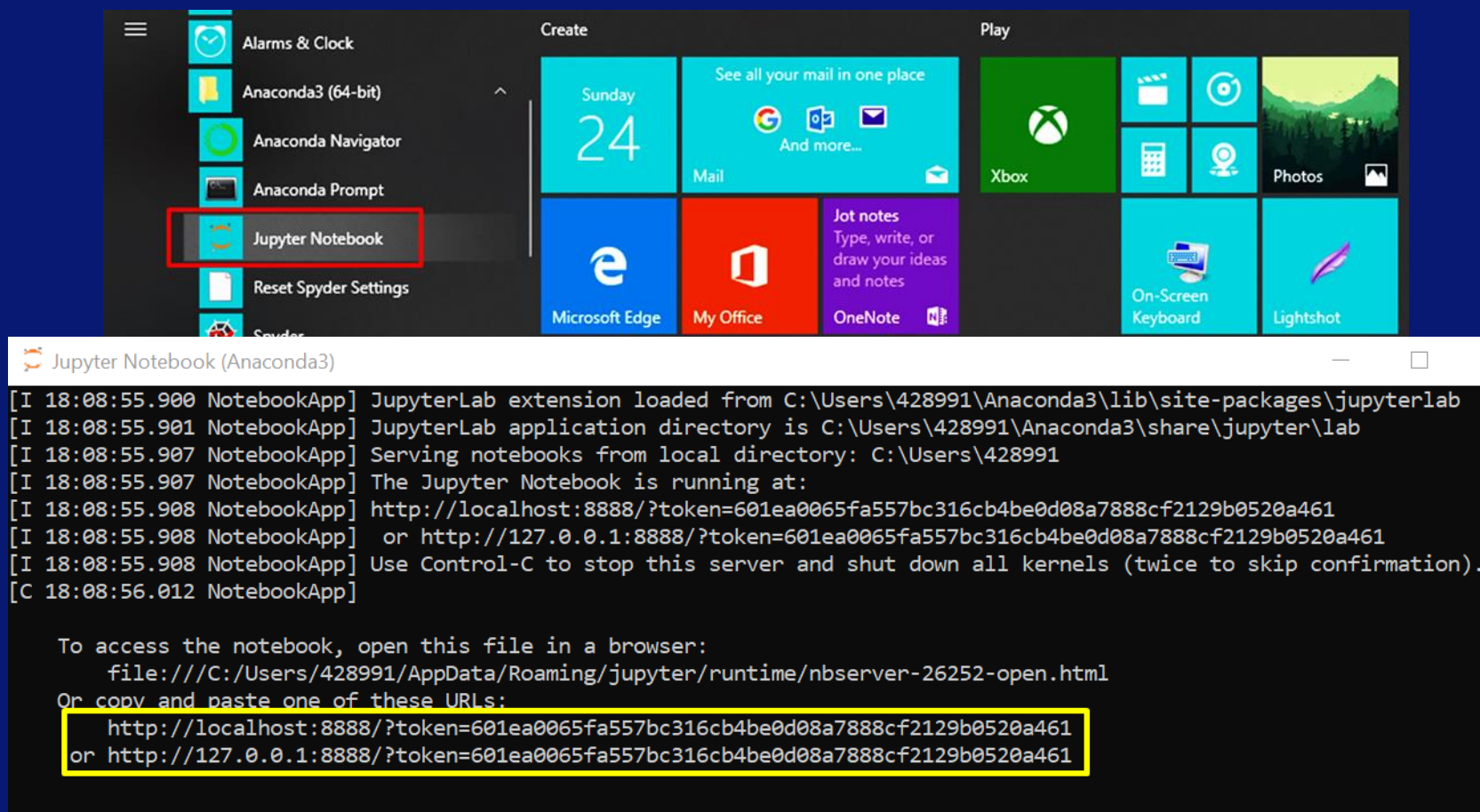




Code Editors and IDEs

Launch Jupyter Notebooks

- Launch Jupyter notebook from start menu in Windows or directly from Anaconda Navigator
- Once launched, paste the URL in a browser to access Jupyter Notebooks. In most cases, browser automatically opens!



The image shows a Windows Start menu with the 'Jupyter Notebook' application highlighted in a red box. Below it, a terminal window titled 'Jupyter Notebook (Anaconda3)' displays the following output:

```
[I 18:08:55.900 NotebookApp] JupyterLab extension loaded from C:\Users\428991\Anaconda3\lib\site-packages\jupyterlab
[I 18:08:55.901 NotebookApp] JupyterLab application directory is C:\Users\428991\Anaconda3\share\jupyter\lab
[I 18:08:55.907 NotebookApp] Serving notebooks from local directory: C:\Users\428991
[I 18:08:55.907 NotebookApp] The Jupyter Notebook is running at:
[I 18:08:55.908 NotebookApp] http://localhost:8888/?token=601ea0065fa557bc316cb4be0d08a7888cf2129b0520a461
[I 18:08:55.908 NotebookApp] or http://127.0.0.1:8888/?token=601ea0065fa557bc316cb4be0d08a7888cf2129b0520a461
[I 18:08:55.908 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 18:08:56.012 NotebookApp]
```

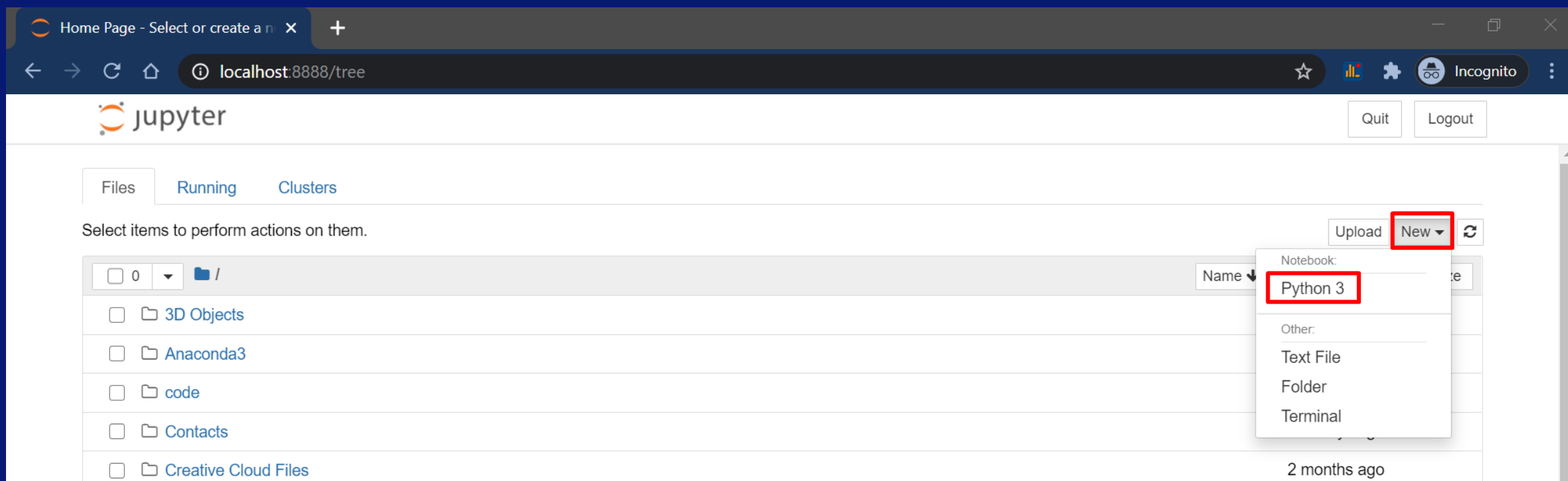
To access the notebook, open this file in a browser:
file:///C:/Users/428991/AppData/Roaming/jupyter/runtime/nbserver-26252-open.html
Or copy and paste one of these URLs:
<http://localhost:8888/?token=601ea0065fa557bc316cb4be0d08a7888cf2129b0520a461>
or <http://127.0.0.1:8888/?token=601ea0065fa557bc316cb4be0d08a7888cf2129b0520a461>



Code Editors and IDEs

Launch Jupyter Notebooks

- When the notebook opens in your browser, you will see the Notebook Dashboard
- Dashboard will show a list of the notebooks, files, and subdirectories from the user's home directory
- Click on **New** and select **Python3** to start a python notebook

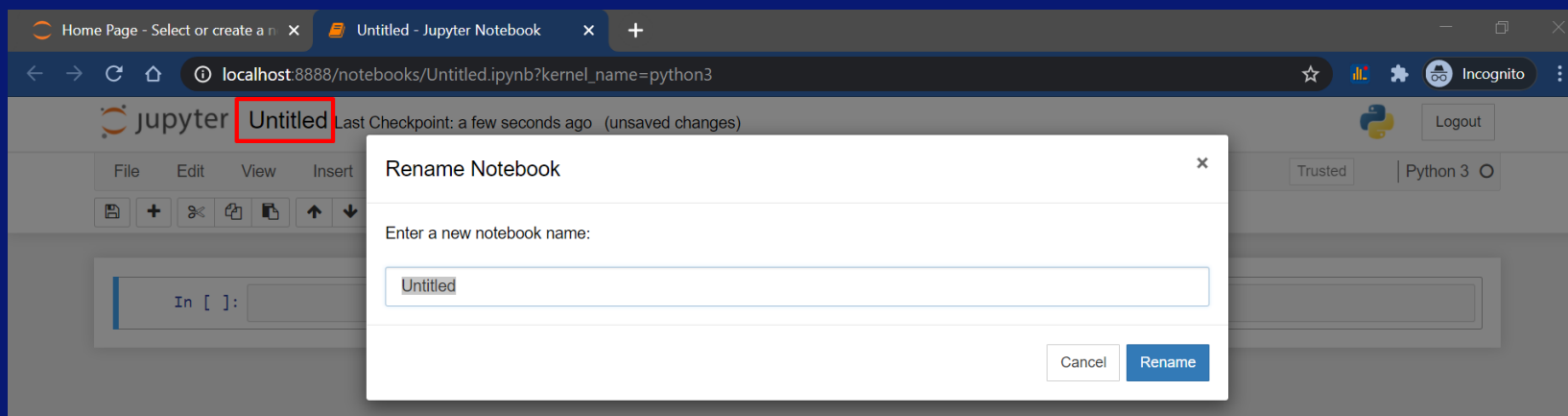




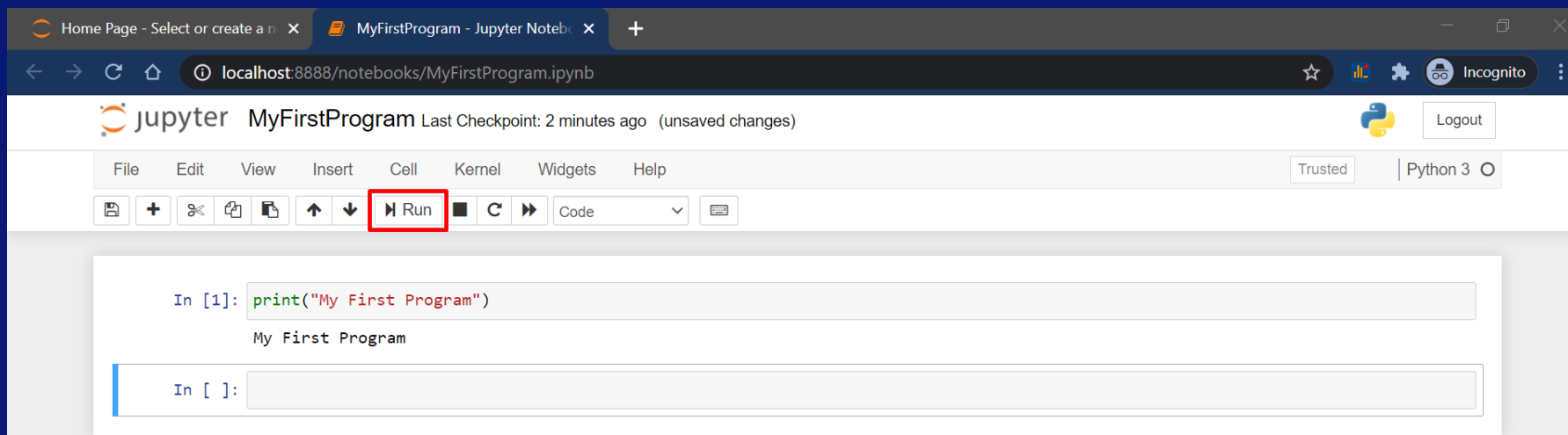
Code Editors and IDEs

Launch Jupyter Notebooks

- Double click on **Untitled** to rename the notebook



- Write a simple python statement and click on **Run** to execute the statement in the cell

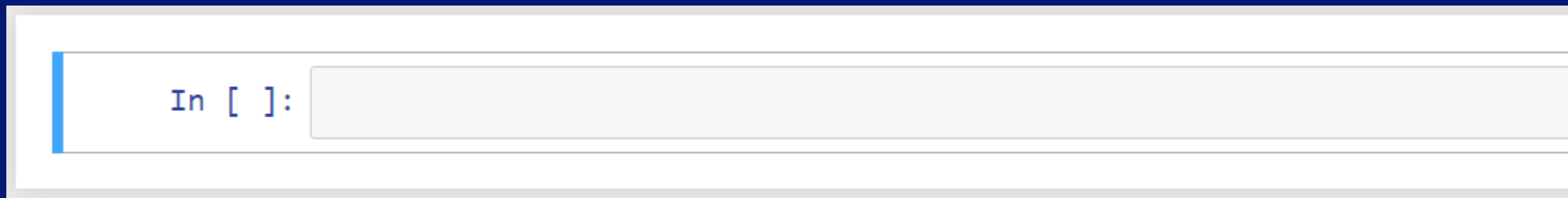


Code Editors and IDEs

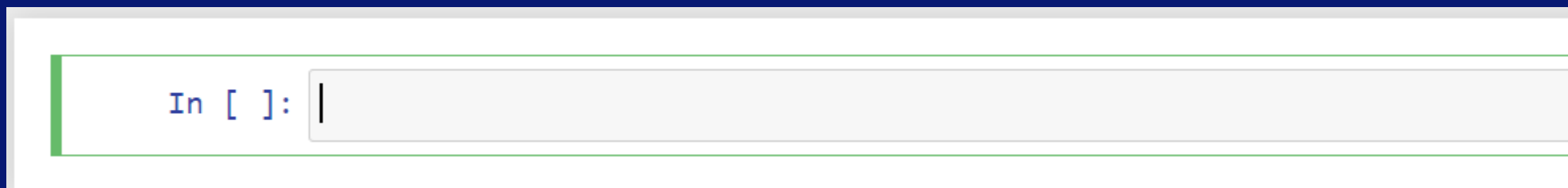
Jupyter Notebooks tips

Command mode vs. Edit mode

- **Command mode** binds the keyboard to notebook level actions. Actions like cell copy, delete, paste can be performed by keyboard keys. Indicated by a grey cell border with a blue left margin



- **Edit mode** when you're typing in a cell. Indicated by a green cell border. Keyboard keys are utilized for typing the instructions. To switch to command mode, click outside the cell active area

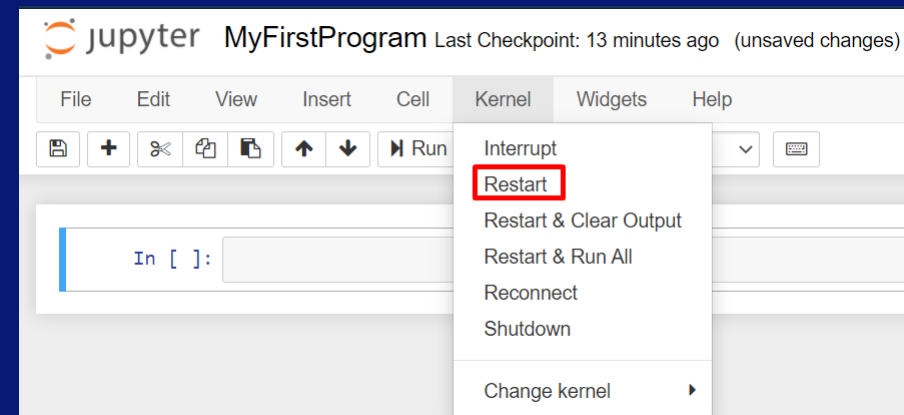


Code Editors and IDEs

Jupyter Notebooks tips

Command mode shortcuts

- Ctrl + enter - run cell and stay in same cell
- Alt + enter - run cell, select below cell in edit mode
- Shift + enter - run cell, select below cell in command mode
- A - insert cell above
- B - insert cell below
- C - copy cell
- V - paste cell
- DD or X - delete selected cell
- Shift + M - merge selected cells, or current cell with cell below if only one cell selected
- II - interrupt kernel, once interrupted restart the kernel as shown
- OO - restart kernel
- M - change cell to markdown mode (good for documentation)

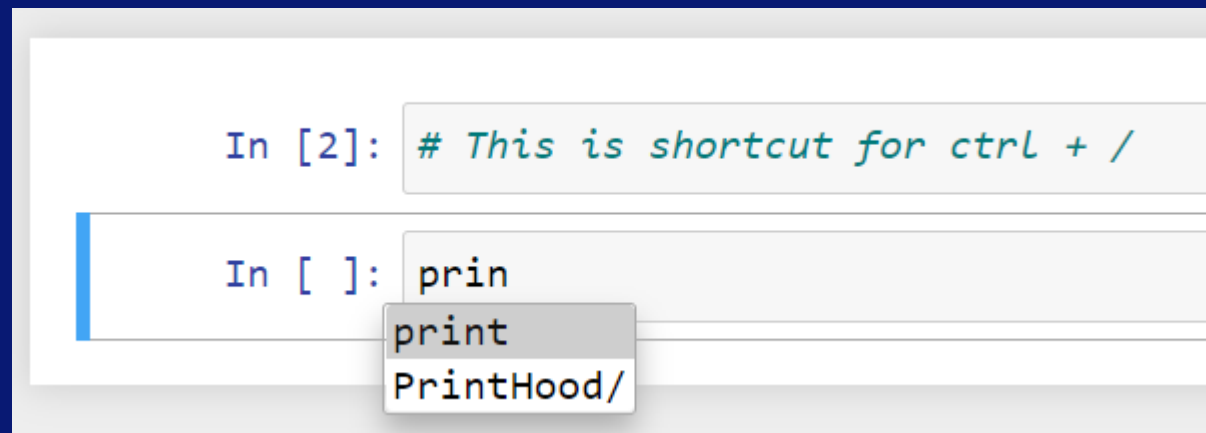


Code Editors and IDEs

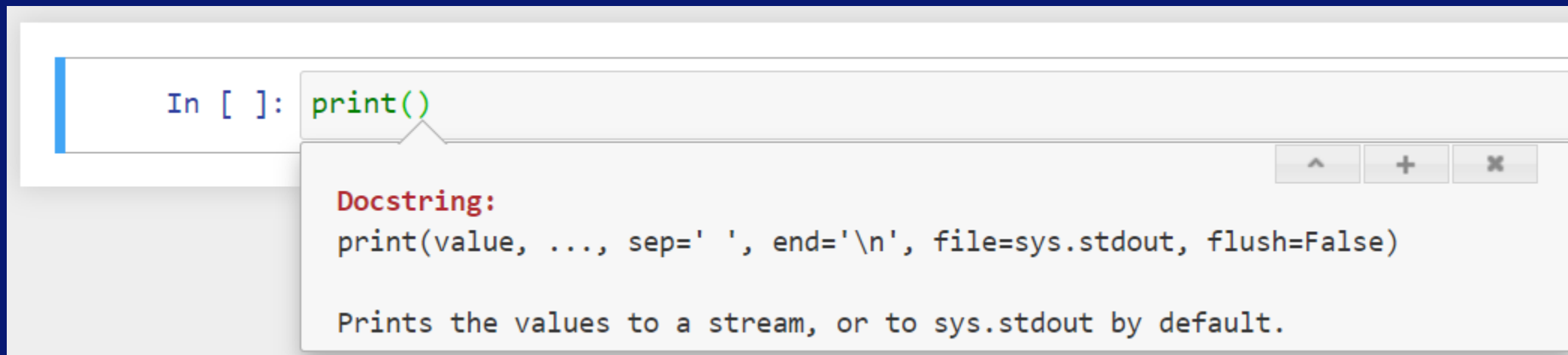
Jupyter Notebooks tips

Edit mode shortcuts

- Ctrl+ / - toggle comment lines
- tab - code completion or indent
- shift + tab - tooltip



A screenshot of a Jupyter Notebook cell in edit mode. The first line of code is `In [2]: # This is shortcut for ctrl + /`. The second line is `In []: prin`. A dropdown menu is visible below the second line, showing suggestions: `print` (highlighted), `print`, and `PrintHood/`.



A screenshot of a Jupyter Notebook cell in edit mode. The first line of code is `In []: print()`. A tooltip is displayed below the code, showing the docstring for the `print()` function. The tooltip text is: **Docstring:**
`print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)`
Prints the values to a stream, or to sys.stdout by default.

Code Editors and IDEs

Jupyter Notebooks tips

Pretty Display of Variables

- By default Jupyter notebooks displays output of only last variable
- Value of multiple statements can be displayed at once by using below command

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

```
In [1]: a = 1
        b = 2
        a
        b

Out[1]: 2

In [2]: from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"

In [3]: a = 1
        b = 2
        a
        b

Out[3]: 1
Out[3]: 2

In [ ]:
```



Code Editors and IDEs

Jupyter Notebooks tips

Easy links to documentation

- Using `?` before a command gives access to the Docstring for quick reference on syntax

Ex: `?print`, `?input`

```
In [4]: ?print
```

```
In [ ]:
```



```
Docstring:  
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)  
  
Prints the values to a stream, or to sys.stdout by default.  
Optional keyword arguments:  
file:  a file-like object (stream); defaults to the current sys.stdout  
sep:   string inserted between values, default a space.  
end:   string appended after the last value, default a newline.  
flush: whether to forcibly flush the stream.  
Type:      builtin_function_or_method
```





Code Editors and IDEs

Jupyter Notebooks tips

Mark down files for documentation

- Write markdown files for easy documentation and sharing
- Click on **Run** to render the file

The image shows the Jupyter Notebook editor interface. The title bar reads "jupyter PythonProgramming (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar contains icons for saving, adding, deleting, copying, pasting, undo, redo, and a dropdown menu currently set to "Markdown". The main editing area contains the following markdown code:

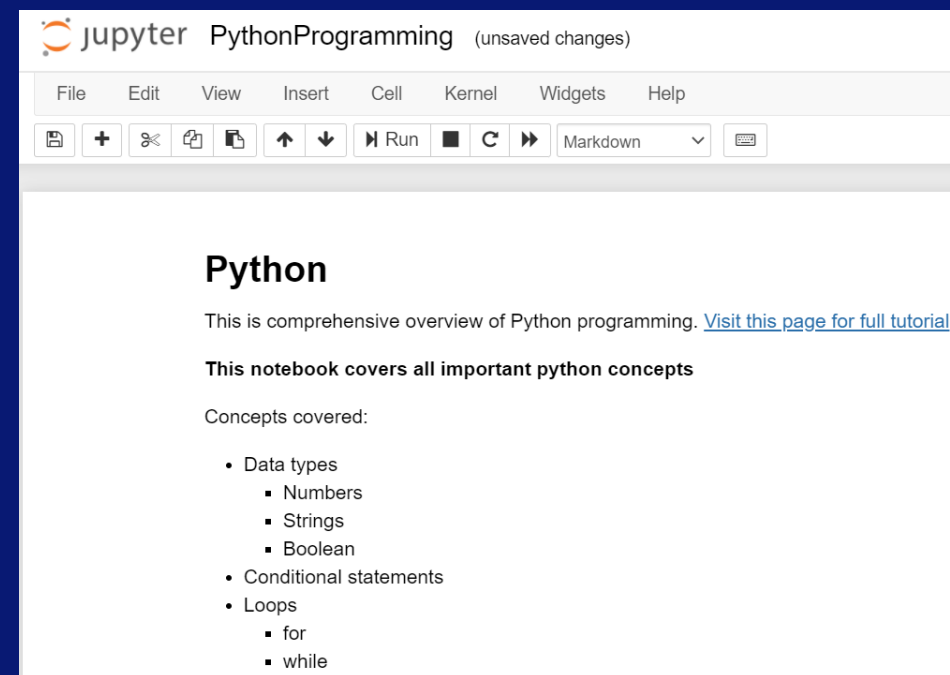
```
# Python

This is comprehensive overview of Python programming.
<a href="https://www.facebook.com/vikram.devops">Visit this page for full tutorial</a>

**This notebook covers all important python concepts**

Concepts covered:

* Data types
  * Numbers
  * Strings
  * Boolean
* Conditional statements
* Loops
  * for
  * while
```





Code Editors and IDEs

Jupyter Notebooks tips

Execution time for code: `%timeit`

- `%timeit` automatically determine the execution time of the single-line Python statement that follows it
- For Multiline Python statements use `%%timeit`

```
In [1]: %timeit squares = [ num for num in range(1000)]  
  
42.5 µs ± 7.64 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

```
In [2]: %%timeit  
squares = []  
for num in range(10000):  
    squares.append(num ** 2)  
  
2.71 ms ± 130 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

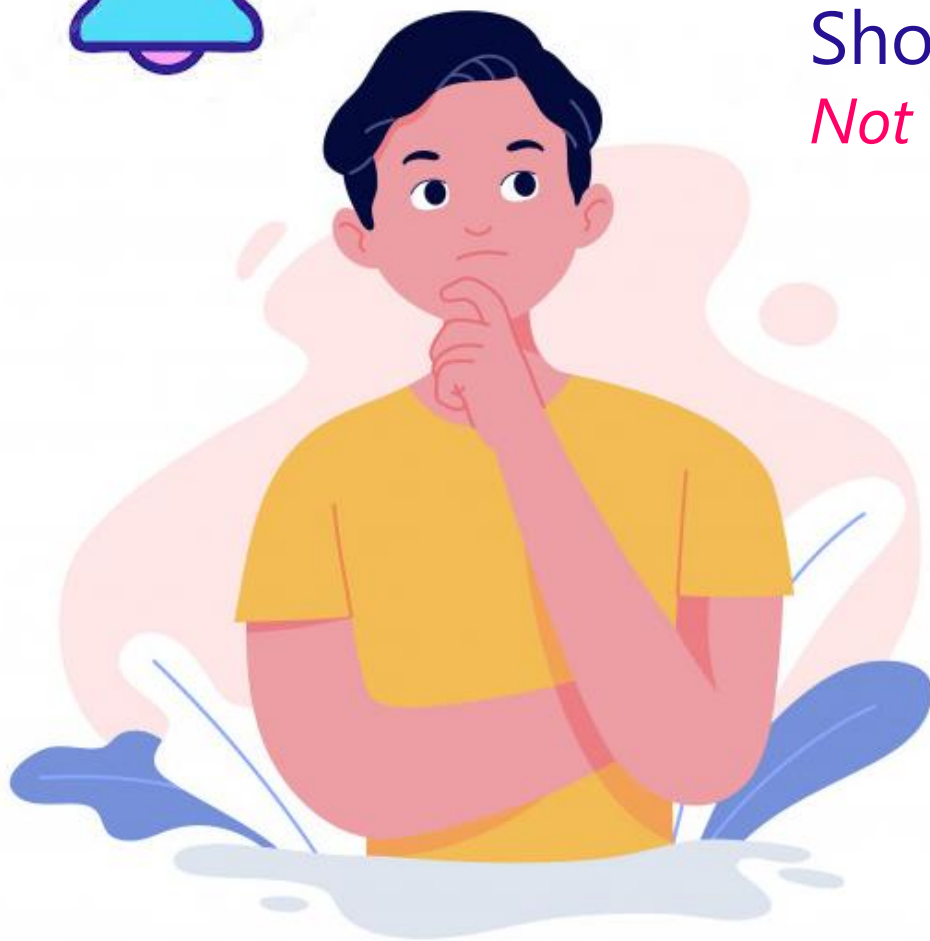
```
In [ ]:
```





Should beginner programmers avoid using IDEs?

Not necessarily



Many text editors with language specific plugins or IDEs support syntax highlighting, auto-complete, or smart refactoring etc., which simplifies code building and speeds up the development.

IDEs are what probably most professional programmers use to write code, and they make programming easier to learn.

However, IDEs do hide things from you. They hide what software is really being used, for example, compiling and running the programs without using an IDE.

So, use an IDE or whatever makes you productive when you are in the business of software development. But try to stay away from them if you are a beginner and you really want to learn what you are doing.



Python

- **Python** is an interpreted, high-level, general-purpose programming language
- It is an object-oriented programming language created by Guido Rossum in 1989. It is free and open-source
- It is used for web development (server-side), software development, mathematics and system scripting.
- It has simple easy-to-use syntax and considered as programmer's first language
- Many large companies use the Python programming language include NASA, Google, YouTube, BitTorrent, etc
- Python is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and Data sciences before of vast support for libraries
- Because python code is automatically compiled to byte code and executed, Python is suitable for use as a scripting language, Web application implementation language, etc
- Most of the Linux distributions have Python pre-installed in it, although it is better to update it to the latest available version





Python

Features of Python

- Built-in high level data types: **strings**, **lists**, **sets**, **tuples**, **dictionaries**, etc.
- The usual control structures: **if**, **if-else**, **if-elif-else**, **while**, plus a powerful collection iterator (**for**)
- Multiple levels of organizational structure: **functions**, **classes**, **modules**, and **packages**
- Compile on the fly to **byte code**. Source code is compiled to byte code without a separate compile step. Source code modules can also be "precompiled" to byte code files
- **Object oriented**: Python provides a consistent way to use objects: everything is an object in Python
- It is dynamically typed and garbage-collected
- Support for wide range of libraries and packages





Python

Zen of Python

- The Zen of Python are guidelines for the design of the Python language. Your Python code doesn't necessarily have to follow these guidelines, but they're good to keep in mind
- Use `import this` to bring the guidelines in the python shell

```
428991@LINL190904638 MINGW64 ~/AppData/Local/Programs/Terminus
$ python
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>> _
```





Python

Compiled vs Interpreted language

Compiled Languages

- When you write a program in C/C++, you have to compile it
- Compilation involves translating your human understandable code to machine understandable code, or Machine Code
- Machine code is the base level form of instructions that can be directly executed by the CPU
- Upon successful compilation, your code generates an executable file
- Executing this file runs the operations in your code step by step

Interpreted Languages

- Python is an interpreted language, although it has a compilation step
- Python code, written in `.py` file is first compiled to what is called `bytecode` and stored in `.pyc` format
- Instead of translating source code to machine code like C++, Python code is translated to `bytecode`
- This bytecode is a low-level set of instructions that can be executed by a Python interpreter
- **Instead of executing the instructions on CPU, bytecode instructions are executed on a Virtual Machine**





Python

PVM (Python Virtual Machine)

- Every programming language converts its code to machine language that is understood by the CPU. This is done by a compiler of that language
- The Python compiler also does the same thing but in a slightly different manner. When we run a Python program, two steps happen:
 1. The code gets converted to **Byte Code**
 2. **Byte Code** gets converted to Machine Code which is understandable by the computer
- The second step is being done by PVM or Python Virtual Memory which is a software/interpreter that converts the byte code to machine code for given operating system
- Python's byte code can't be understood by CPU. So we need actually an interpreter called the python virtual machine to executes the byte codes
- So, PVM is also called as Python Interpreter
- That's the reason, while delivering python projects, ***.pyc** files are given with PVM so that users can see the output directly

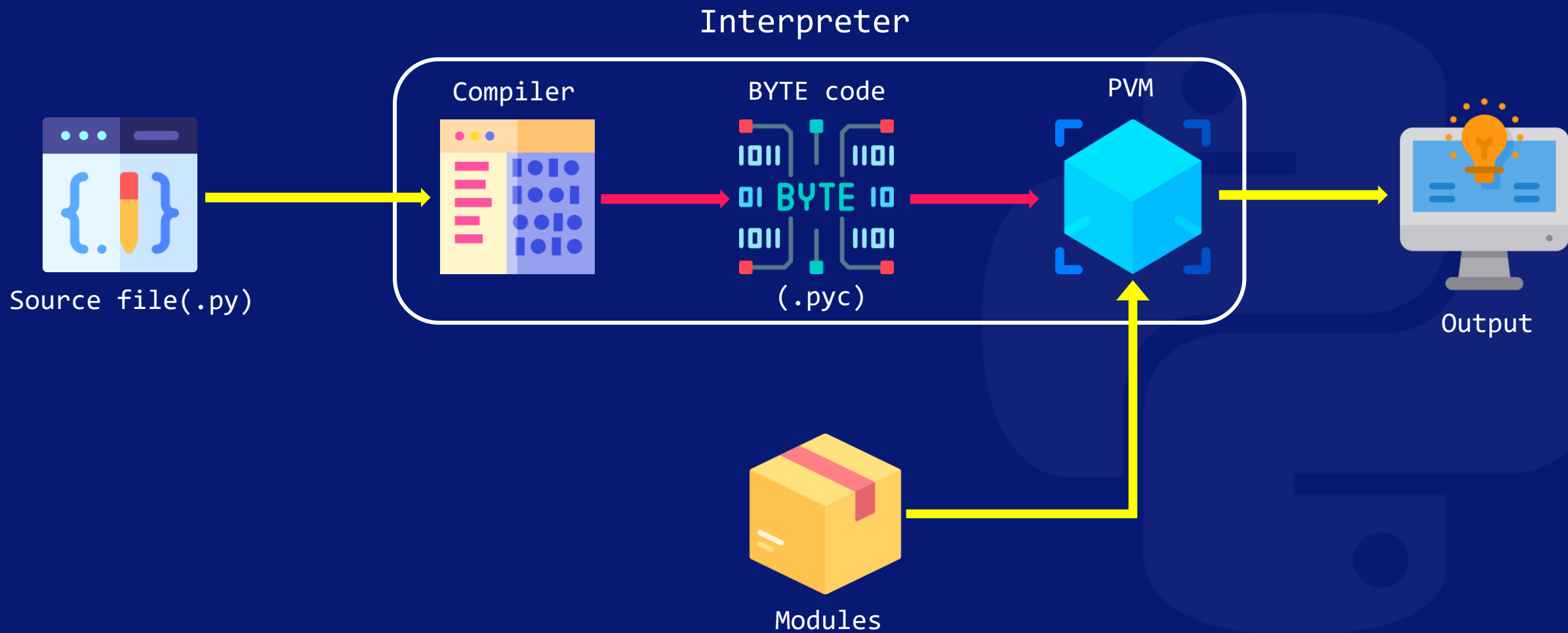




Python

PVM (Python Virtual Machine)

Program Execution Flow



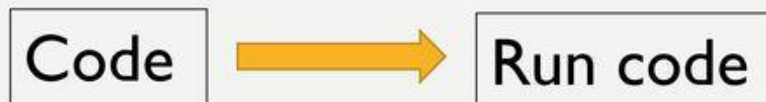
Compiled vs Interpreted language

COMPILED VS INTERPRETED LANGUAGES

- ❑ A compiled language is when a person writes the code the compiler separates the file and the end result is an executable file. Basically, the **owner keeps the source code**.



- ❑ Interpreted languages are different because the code is not compiled first hand. Instead, a copy is given to another machine and that machine interprets it. An example is **JavaScript** and **PHP** (which is used everywhere online).



Compiled		Interpreted	
PROS	CONS	PROS	CONS
ready to run	not cross platform	cross-platform	interpreter required
often faster	inflexible	simpler to test	often slower
source code is private	extra step	easier to debug	source code is public



Python

Interpreted language

Advantages

- One advantage of interpreted languages is that they are platform-independent
- As long as the Python bytecode and the Virtual Machine have the same version, Python bytecode can be executed on any platform (Windows, MacOS, etc)

Disadvantages

- Python is often accused of being **slow**
- It is because the interpreter has to do extra work to have the bytecode instruction translated into a form that can be executed on the machine





Python

Dynamic Typing

- Python is a dynamically typed language. This means that the Python interpreter does type checking only as code runs, and the type of a variable is allowed to change over its lifetime
- We don't have to declare the type of variable while assigning a value to a variable in Python
- In static-typed languages like C, C++, Java, etc., there is a strict declaration of variables before assigning values to them and validated as your program is compiled
- Dynamic typing provides a lot of freedom, but simultaneously it makes your code risky and sometimes difficult to debug

```
>>> if False:
...     1 + "two" # This line never runs, so no TypeError is raised
... else:
...     1 + 2
...
3
>>> 1 + "two" # Now this is type checked
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Python

```
>>> thing = "Hello"
>>> type(thing)
<class 'str'>

>>> thing = 28.1
>>> type(thing)
<class 'float'>
```

Python



Python

Python Garbage Collection

Memory Management

- A programming language uses objects in its programs to perform operations. Objects include simple variables, like strings, integers, or Booleans. They also include more complex data structures like Arrays, Lists, Hashes, or Classes
- The values of the program's objects are stored in memory for quick access
- In many programming languages, a variable in your program code is simply a pointer to the address of the object in memory
- When a variable is used in a program, the process will read the value from memory and operate on it
- In early programming languages, developers were responsible for all memory management in their programs. This means before creating an object, you are first needed to allocate the memory for your object. After you are done with the object, you then needed to deallocate it to free that memory for other users
- This led to two problems:
 1. **Forgetting to free your memory:** If you don't free your memory when you're done using it, it can result in memory leaks. This can lead to your program using too much memory over time. For long-running applications, this can cause serious problems
 2. **Freeing your memory too soon:** The second type of problem consists of freeing your memory while it's still in use. This can cause your program to crash if it tries to access a value in memory that doesn't exist, or it can corrupt your data. A variable that refers to memory that has been freed is called a **dangling pointer**

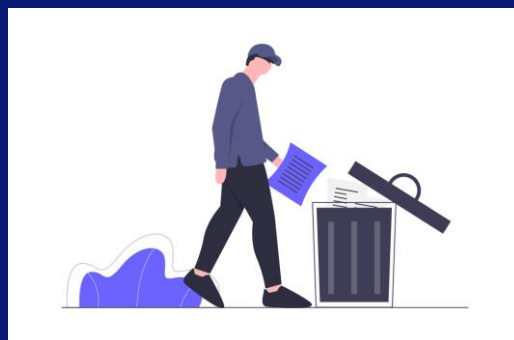


Python

Python Garbage Collection

Automatic memory management

- With automatic memory management/garbage collection, programmers no longer needed to manage memory themselves. Rather, the runtime handled this for them
- There are a few different methods for automatic memory management, but the popular one is **reference counting**
- With reference counting, the runtime keeps track of all of the references to an object. When an object has zero references to it (program no longer use this object), the object gets deleted
- For programmers, automatic memory management adds a number of benefits. It's faster to develop programs without thinking about low-level memory details. Further, it can help avoid costly memory leaks or dangerous dangling pointers
- The only overhead in using automatic memory management is keeping track all of all references





Python

Python Garbage Collection

How Python implements GC

- Python's Interpreter can be implemented in C language – Cpython or in Java – Jpython, although Cpython is widely used implementation
- To see which Python you're using, run the following command in your terminal
- `python -c 'import platform; print(platform.python_implementation())'`

```
428991@LINL190904638 MINGW64 ~/AppData/Local/Programs/Terminus
$ python -c 'import platform; print(platform.python_implementation())'
CPython
```

- The main garbage collection mechanism in CPython is through reference counts. Whenever you create an object in Python, the underlying C object has both a Python type (such as list, dict, or function) and a reference count
- The object's reference count is incremented whenever the object is referenced, and it's decremented when an object is dereferenced. If an object's reference count is 0, the memory for the object is deallocated



Python

Python Garbage Collection

Checking the reference counts

- To check the reference counts, use `sys` module from the Python standard library

```
import sys
sys.getrefcount(object-name)
```

```
428991@LINL190904638 MINGW64 ~/AppData/Local/Programs/Terminus
$ python
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> a = "hello-world"
>>> sys.getrefcount(a)
2
>>> _
```



There are two references to the variable `a`. One is from creating the variable. The second is when we pass the variable `a` to the `sys.getrefcount()` function.





Python

Running Python Programs

- Python is interpreted language rather than compiled
- Python code is executed line by line, which allows programming to be interactive in a way that is not directly possible with compiled languages like Fortran, C, or Java
- The engine that translates and runs Python is called the Python Interpreter
- There are several ways to run Python code
 1. Immediate Mode using Python interpreter
 2. Script mode in IDE
 3. Interactive mode in Jupyter notebook

Immediate Mode

- Once Python is installed, type `python` in the command line to invoke the interpreter in immediate mode
- We can directly type in Python code, and press Enter to get the output
- The `>>>` is called the Python prompt. The interpreter uses the prompt to indicate that it is ready for instructions
- To exit this mode, type `quit()` and press enter





Python

Running Python Programs

Immediate Mode



You can also pass a string of commands directly to the interpreter. Observe the need to escape double quotes.

```
428991@LINL190904638 MINGW64 ~/AppData/Local/Programs/Terminus
$ python --version
Python 3.9.1

428991@LINL190904638 MINGW64 ~/AppData/Local/Programs/Terminus
$ python
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
>>> quit()

428991@LINL190904638 MINGW64 ~/AppData/Local/Programs/Terminus
$ _
```

```
428991@LINL190904638 MINGW64 ~/AppData/Local/Programs/Terminus
$ python -c "print('Hello World')"
Hello World

428991@LINL190904638 MINGW64 ~/AppData/Local/Programs/Terminus
$ python -c "print(\"Hello World\")"
Hello World

428991@LINL190904638 MINGW64 ~/AppData/Local/Programs/Terminus
$ python -c "a=1;b=2;print(a,b)"
1 2
```



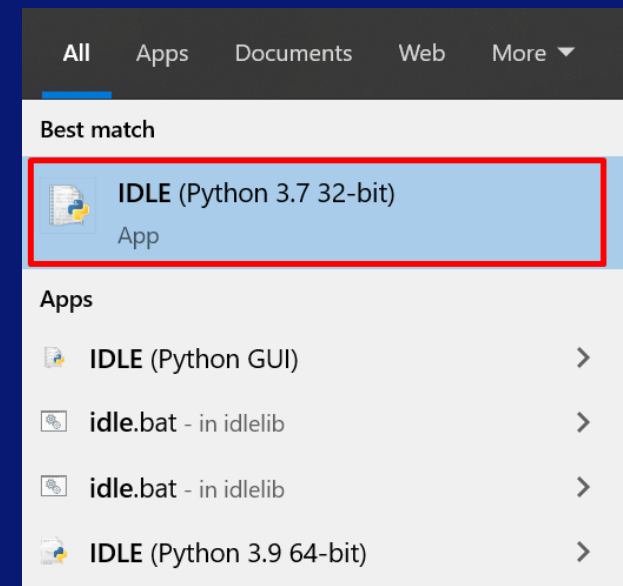


Python

Running Python Programs

Script Mode

- All Python script files should have `.py` extension. We can use any text editor to create these files or have the IDE save the files in this format
- By default, with Python installation, an IDE named IDLE is also installed
- When you open IDLE, an interactive Python Shell is opened by default, which is similar to Immediate Mode as seen earlier
- Search for IDLE in the Windows apps and open



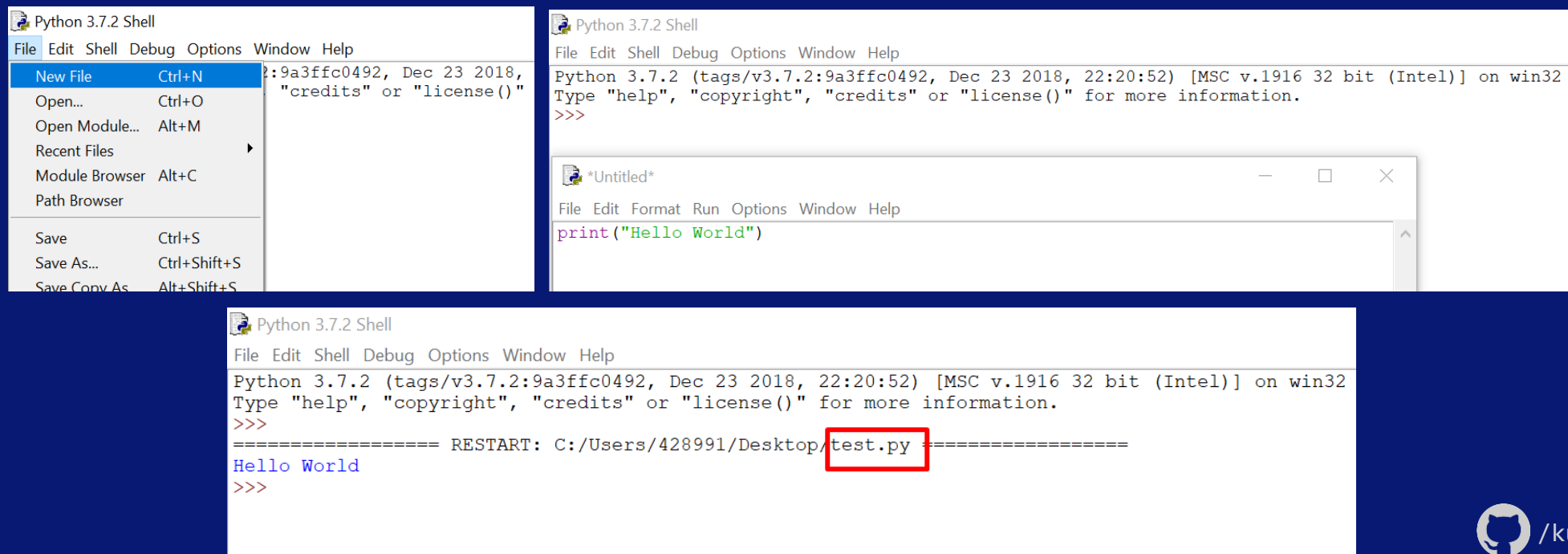


Python

Running Python Programs

Script Mode

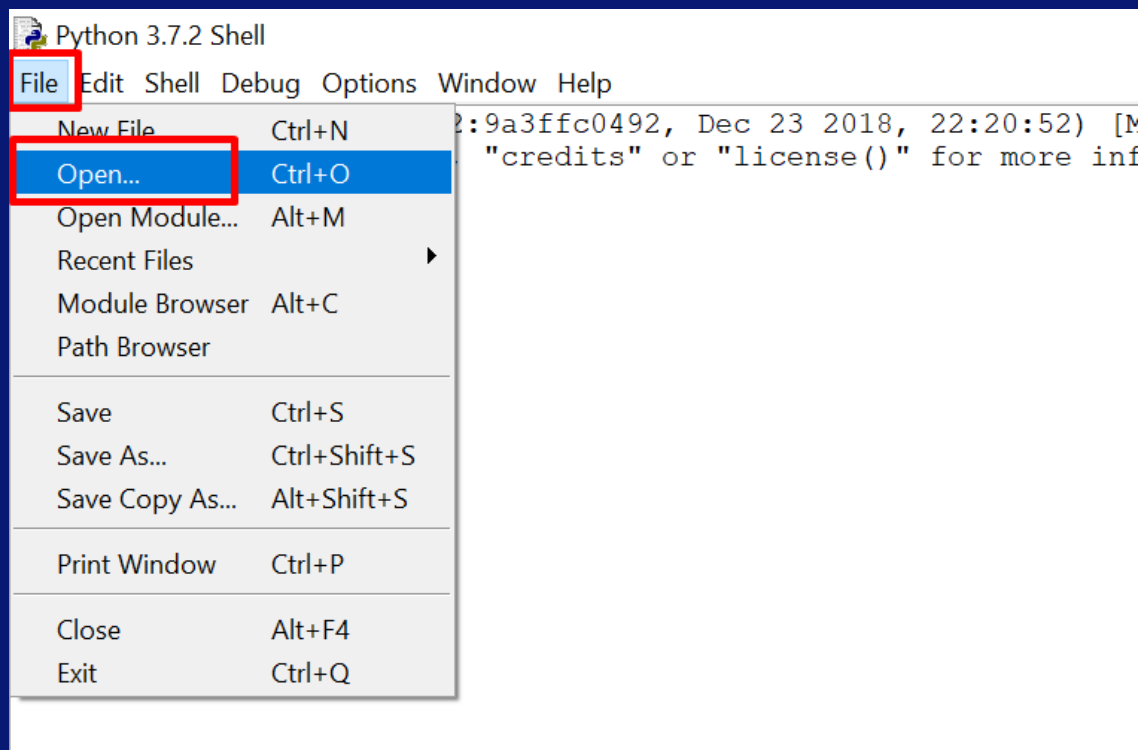
- Once the interactive shell is opened, click on File -> New File. A new editor window opens
- Type `print("Hello World")` and click on Run -> Run Module. The IDE prompts to save the file
- Save it to a desired location by specifying a filename without any extensions
- Observe that file automatically saves with `.py` extension. Ex: `test.py`
- Once the code runs, its output is shown in the interactive terminal



Python

Opening py files

- You can open python files stored in your local filesystem directly from IDLE
- Open IDLE, click on File -> Open
- Navigate to the directory where the `.py` file is stored and open it

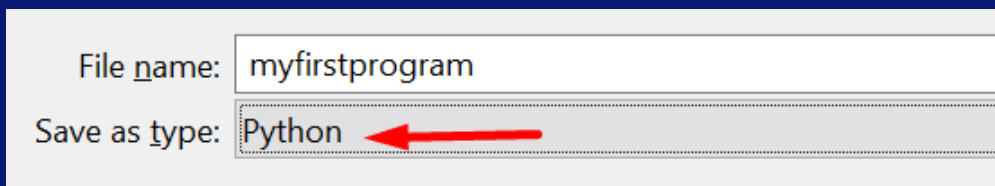




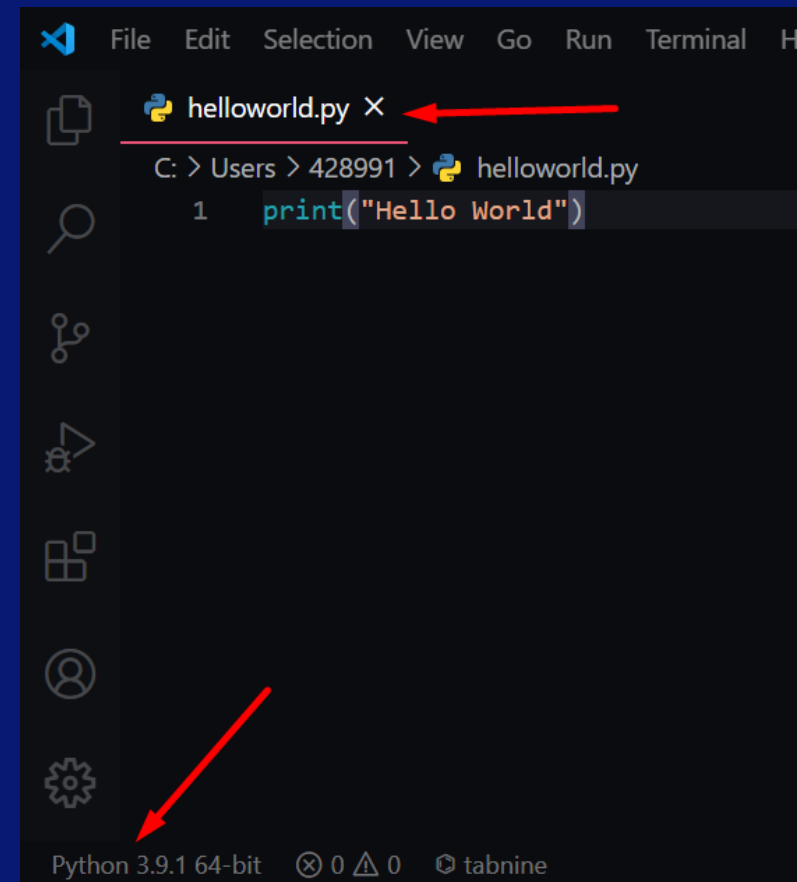
Python

Using VS code to create/run py files

- Open VS Code
- Click on File -> New File. An untitled file is opened by default
- Write the required code and save it by pressing ctrl + s
- Save it to a desired location by specifying a filename without any extensions. Make sure the Save as type is selected as **Python**
- File gets automatically saved with **.py** extension



- Observe that as soon as VS Code detects the python file, it automatically selects the appropriate Python Interpreter
- If there is any issue with the interpreter or interpreter not being set, refer <https://code.visualstudio.com/docs/python/python-tutorial#prerequisites>





Python

Using VS code to create/run py files

- Run the program by selecting the Run button on the top right corner
- VS Code automatically invokes the interpreter and runs the Python program
- Terminal shows the location of the interpreter executable, the location of the python file being run and the output of the program

The screenshot shows the Visual Studio Code interface with a file named `helloworld.py` open. The code in the editor is `print("Hello World")`. A red arrow points to the Run button (a green play icon) in the top right corner of the editor. Below the editor, the TERMINAL panel is active, showing the command `PS C:\Users\428991> & "C:/Program Files/Python39/python.exe" c:/Users/428991/helloworld.py` and the output `Hello World`. A red arrow points to the output text. The bottom right corner of the image contains a GitHub logo and the username `/kunchalavikram1427`.

```
File Edit Selection View Go Run Terminal Help helloworld.py - Visual Studio Code
```

helloworld.py X

```
C: > Users > 428991 > helloworld.py
1 print("Hello World")
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE 1: Python

```
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\428991> & "C:/Program Files/Python39/python.exe" c:/Users/428991/helloworld.py
Hello World
PS C:\Users\428991>
```


/kunchalavikram1427

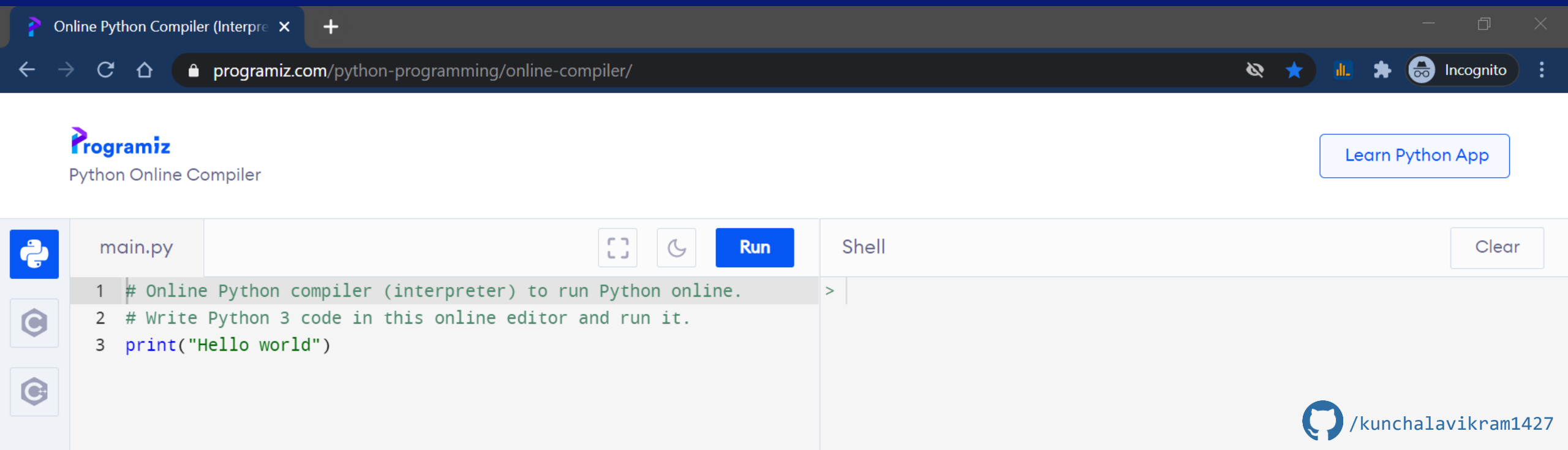


Python

Online Compilers

- You can run Python programs in the browser using online Python interpreters like
 - <https://www.programiz.com/python-programming/online-compiler/>
 - <https://repl.it/> - One of the best online python interpreter

 https://www.youtube.com/watch?v=un83OvQS00o&ab_channel=SanjinDedic



The screenshot shows the Programiz Python Online Compiler interface. The browser tab is titled "Online Python Compiler (Interpreter)". The address bar shows the URL "programiz.com/python-programming/online-compiler/". The page header includes the Programiz logo and the text "Python Online Compiler", along with a "Learn Python App" button. The main area is divided into two panels. The left panel, titled "main.py", contains the following Python code:

```
1 # Online Python compiler (interpreter) to run Python online.
2 # Write Python 3 code in this online editor and run it.
3 print("Hello world")
```

The right panel, titled "Shell", is currently empty and shows a prompt character ">". A "Run" button is located between the code editor and the shell. A "Clear" button is located in the top right corner of the shell panel. The bottom right corner of the image shows a GitHub logo and the username "/kunchalavikram1427".



PART - 02

Introduction to Python Python Language Syntax





Python

Comments in Python

Single-Line Comments

- As programs get bigger and more complicated, they get more difficult to read
- For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing
- A comment in a program is text that is intended only for the human reader — it is completely ignored by the interpreter
- In Python, the `#` token starts a single line comment. The rest of the line is ignored

```
# printing a string                                comments
print("Welcome to DevOps Made Easy Tutorials")

salestax = 1.10      # defining a sales tax of 10%

# define global variable store the sum value
sum = 0
```



Python

Comments in Python

Single-Line Comments

```
# -----comments
# demonstrates how to write ms excel files using python-openpyxl
#
# (C) 2015 Frank Hofmann, Berlin, Germany
# Released under GNU Public License (GPL)
# email frank.hofmann@efho.de
# -----

# define the general structure of the product with default values
product = {
    "productId": 0,          # product reference id, default: 0
    "description": "",       # item description, default: empty
    "categoryId": 0,         # item category, default: 0
    "price": 0.00            # price, default: 0.00
}
```





Python

Comments in Python

Multi-Line Comments

- Python doesn't specifically offer a way to write multiline comments. We can use # to comment all the lines or wrap them up within 3 single quotes before and after the comment
- Technically, this is not a comment but a string, but the computer still ignores it, so we will use it

```
# THIS IS A MULTILINE  
# COMMENT IN PYTHON  
# USING THE '#'  
print("Welcome to DevOps Made Easy Tutorials")  
  
...  
  
THIS IS A MULTILINE COMMENT  
USING STRINGS  
WRAP THE COMMENT IN 3 SINGLE QUOTES  
...  
  
sum = 0
```

comments

Python

Comments in Python

Docstrings

- Python has a built-in concept called docstrings, which is a great way to associate documentation with Python modules, functions, classes, and methods. A docstring is added as a comment right below the function, module, or object head, and describes what the function, module, or object does. It is expected to follow these rules:
 - A docstring is either a single line, or a multi-line comment
 - Begin the docstring with a capital letter, and end it with a period
- Access the docstring using `__doc__` attribute

```
def add(value1, value2):  
    """Calculate the sum of value1 and value2."""  
    return value1 + value2  
print(add.__doc__)
```

```
>>> Calculate the sum of value1 and value2.
```

Docstrings

docstrings.py X

```
docstrings.py > ...  
1  # Docstrings example  
2  def add(a, b):  
3      """  
4          Description:  
5          add method  
6          Parametes:  
7          a:accepts int,float  
8          b:accepts int,float  
9          Returns:  
10         This function returns the  
11         sum of arguments passed.  
12         """  
13     return a + b  
14 print(add.__doc__)  
15
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
PS D:\> & "C:/Program Files/Python39/python.exe" "c:/Users/428991/OneDrive - Alstom/Study/Python/Programs/docstrings.py"
```

```
Description:  
add method  
Parametes:  
a:accepts int,float  
b:accepts int,float  
Returns:  
This function returns the  
sum of arguments passed.
```



/kunchalavikram1427



Python

Comments in Python

Docstrings

- Doc strings can be viewed with several tools, e.g. `help()`, `obj.__doc__`, and, in IPython, a question mark (?) after a name will produce help
- There are also tools that extract and format doc strings, for example:
 - **pydoc** - Documentation generator and online help system (<http://docs.python.org/lib/modulepydoc.html>)
 - **epydoc** - Automatic API Documentation Generation for Python (<http://epydoc.sourceforge.net/index.html>)

```
$ python
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys.stdout.
    sep: string inserted between values, default a space.
    end: string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
In [4]: ?print

In [ ]:
```

Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type: builtin_function_or_method

Ex: `?print`, `?input`



Python

Lines

- **End-of-Line** terminates a statement in Python (semicolon for C)
- Longer statements can span to multiple lines with the use of backslash as last character
- Parenthesis, square bracket, or curly bracket can also be used instead of \
- Triple quoted strings can even span multiple lines
- Python supports more than one statement on the same line if statements are separated by ;
- **It is considered to be a bad practice having multiple statements in a single line**

```
message = """ This message will
span several
lines. """
```

```
EOL.py X
EOL.py > ...
1  # Multiline statements
2  sum1 = 1 + 2 + 3+ \
3      + 4 + 5+ \
4      6
5  sum2 = (7 + 8 + 9+
6      + 10 + 11+
7      12)
8
9  print(sum1, sum2)
10
11 # Multi statements in a linear manner
12 a = 1 ; b = 2 ; c = 3 ; d = 4
13 print(a, b, c, d)

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

PS D:\> & "C:/Program Files/Python39/python.exe" "c:/Users/428991/OneDrive - Alstom/Study/Python/Programs/EOL.py"
21 57
1 2 3 4
PS D:\> 
```



Python

Indentation

- Python uses indentation to define **control** and **loop** constructs. This contributes to Python's code readability and reduces inconsistency
- The recommended indentation is 4 spaces but tabs or spaces can be used as long as they are consistent. These default spaces for a indentation can be adjusted in the IDE
- However, a close attention is needed to use whitespaces in the program. Unnecessary white spaces leads to **IndentationError**. **Do not mix tabs and spaces!**
- Python uses the colon symbol (:) and indentation for showing where blocks of code begin and end, unlike c which uses { }

```
// C code
for(int i=0; i<10; i++)
{
// curly braces indicate code block
printf("i=%d \n",i);
total += i;
}
printf("Total = %d,total);
```

```
# python
total = 0
for i in range(10):
    # indentation indicates code block
    print("i=", i)
    total += i
print("Total=",total)
```



Python

Indentation

- In Python, indented code blocks are always preceded by a colon (:) on the previous line
- In the snippet on the bottom left, `print("Total=",total)` is in the indented block, and will be executed every time `i` is incremented
- In the snippet on the bottom right, `print("Total=",total)` is outside the block, and will be executed only after `for` loop is executed completely

```
# python
total = 0
for i in range(10):
    # indentation indicates code block
    print("i=", i)
    total += i
    print("Total=",total)
```

```
# python
total = 0
for i in range(10):
    # indentation indicates code block
    print("i=", i)
    total += i
print("Total=",total)
```



Python

Keywords

- Keywords are the **reserved** words in Python and cannot be used to name a variable, function or any other identifier
- They are used to define the syntax and structure of the Python language
- Keywords are case sensitive. There are 36 keywords in Python 3.9
- All the keywords except **True**, **False** and **None** are in lowercase

```
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from',
'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield']
>>> print(len(keyword.kwlist))
36
>>>
```



len function is used to find the length of a list, covered in the later part of the series along with the keywords mentioned above





Python

Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another

Rules for naming identifiers

- Identifier names can have characters: a-z, A-Z, 0-9, underscore, and must begin with a letter or underscore
`age`, `_age`, `my_age`, `myAge` all are valid names, while `1age` is invalid
- Special characters like `!`, `@`, `#`, `$`, `%` are not allowed
- Identifier are case sensitive. So `Age` and `age` are two different identifiers
- Identifiers can be of unlimited length
- Special names, customizing, etc. usually begin and end in double underscores
- Special name classes use single and double underscores (`__init__`)
- Single leading single underscore suggests a "private" method or variable name (`_myage`) and Not imported by `from module import *`
- Modules and packages use all lower case (`import math`)
- Globals and constants use upper case (`PI = 3.14`)
- Classes use camelCase with initial upper (`class MyNewClass`)
- Methods and functions All lower case with words separated by underscores (`def greet_employee(self)`)
- Local variables use lower case or camelCase with initial lower



UP NEXT

Variables, Data types & Operators

References

- <https://wiki.python.org/moin/BeginnersGuide/Programmers>
- [https://code.visualstudio.com/docs/python/python-tutorial# prerequisites](https://code.visualstudio.com/docs/python/python-tutorial#prerequisites)
- <https://cloudacademy.com/blog/python-what-is-it-and-why-is-it-so-popular/>
- <http://pythontutor.com/>





Subscribe to my Facebook page:

<https://www.facebook.com/vikram.devops>

and join my group:

<https://www.facebook.com/groups/171043094400359>

for all latest updates on DevOps, Python and IoT



<https://www.youtube.com/channel/UCE1cGZfooxT7-VbqVbuKjMg>

Q&A

