Programming for Artificial Intelligence

Anab Batool Kazmi

Artificial Intelligence

- Artificial intelligence (AI) is the ***ability of machines to perform tasks that are typically associated with human intelligence***, such as learning and problem-solving.
- The goal of AI is to create ***intelligent systems*** that can ***reason***, ***learn from data, adapt to changing circumstances, and perform tasks with a high degree of autonomy***.
- AI research and development continue to advance rapidly, with ***AI systems becoming increasingly integrated*** into our daily lives and contributing to technological innovation.

Applications of Artificial Intelligence

- web search engines (e.g., Google Search)
- recommendation systems (used by YouTube, Amazon, and Netflix)
- understanding human speech (such as Siri and Alexa)
- self-driving cars (e.g., Waymo)
- generative or creative tools (ChatGPT and AI art)
- competing at the highest level in strategic games (such as chess and Go)

Key elements of AI

- Machine Learning
- Natural Language Processing (NLP)
- Computer Vision
- Expert Systems
- Robotics
- Deep Learning
- Reinforcement Learning

Programming

- Programming is the ***implementation of logic*** to facilitate specified computing operations and functionality. It occurs in one or more ***languages***, which differ ***by application***, ***domain*** and ***programming model***.

Programming Languages differ by Application

Programming languages can vary based on the specific application or purpose for which they are designed. For example:

- ***Web Development:*** Languages like ***HTML, CSS, JavaScript, and PHP*** are commonly used for creating websites.
- ***Data Science:*** Languages such as ***Python and R*** are popular for data analysis and machine learning.
- ***Embedded Systems***: ***C and C++*** are often used to program microcontrollers and embedded devices.
- ***Game Development***: Game developers commonly use languages like ***C# and C++*** for creating video games.

Programming Languages differ by Domain

Programming languages are tailored to different domains or industries. This means that certain languages are better suited for specific tasks within those domains. For instance:

- ***Scientific Computing***: Languages like ***Python and MATLAB*** are preferred for scientific research and simulations.
- ***Financial Software***: Languages such as ***Java and C#*** are used to develop applications for banking and financial services.
- ***Artificial Intelligence***: ***Python*** is widely used for AI and machine learning applications.
- ***Real-Time Systems***: ***C and Ada*** are often used in industries like aerospace and automotive for real-time control systems.

Programming Languages differ by Programming Model

Programming languages also vary in terms of their programming paradigms or models. These models dictate how code is organized and executed. Examples include:

- ***Procedural Programming***: Focuses on procedures or functions that execute sequentially (e.g., ***C***).
- ***Object-Oriented Programming (OOP***): Organizes code into objects with attributes and methods (e.g., ***Java, Python***).
- ***Functional Programming (FP):*** Emphasizes functions as first-class citizens and avoids mutable data (e.g., ***Haskell, Lisp***).
- ***Event-Driven Programming***: Handles events and responses to user or system actions (e.g., ***JavaScript*** for web development).

Programming for Artificial Intelligence

- Programming for Artificial Intelligence (AI) involves ***creating software*** and algorithms ***that enable machines, often computers, to perform tasks that typically require human intelligence***.

Best Artificial Intelligence Programming Languages

An introduction to coding with Python

Python

- Python is ***an interpreted, object-oriented, high-level programming language*** with ***dynamic semantics*** developed by Guido van Rossum. It was originally released in 1991.

Language properties

- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping
- Operator overloading
- Indentation for block structure

Why to Use Python?

- **Easy-to-learn:** Python has relatively few keywords, simple structure, and a clearly defined syntax.
- **Easy-to-read:** Python code is much more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's success is that its source code is fairly easy-to-maintain.
- **A broad standard library:** One of Python's greatest strengths is the bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Support for an interactive mode in which you can enter results from a terminal right to the language, allowing interactive testing and debugging of snippets of code.

Why to Use Python? (cont'd)

- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Where to use Python?

- System management (i.e., scripting)
- Graphic User Interface (GUI)
- Internet programming
- Database (DB) programming
- Text data processing
- Distributed processing
- Numerical operations
- Graphics
- And so on…

Python vs. Java

- Code 5-10 times more concise
- Dynamic typing
- Much quicker development
  - no compilation phase
  - less typing
- Yes, it runs slower
  - but development is so much faster!
- Similar (but more so) for C/C++
- Use Python with Java: JPython!

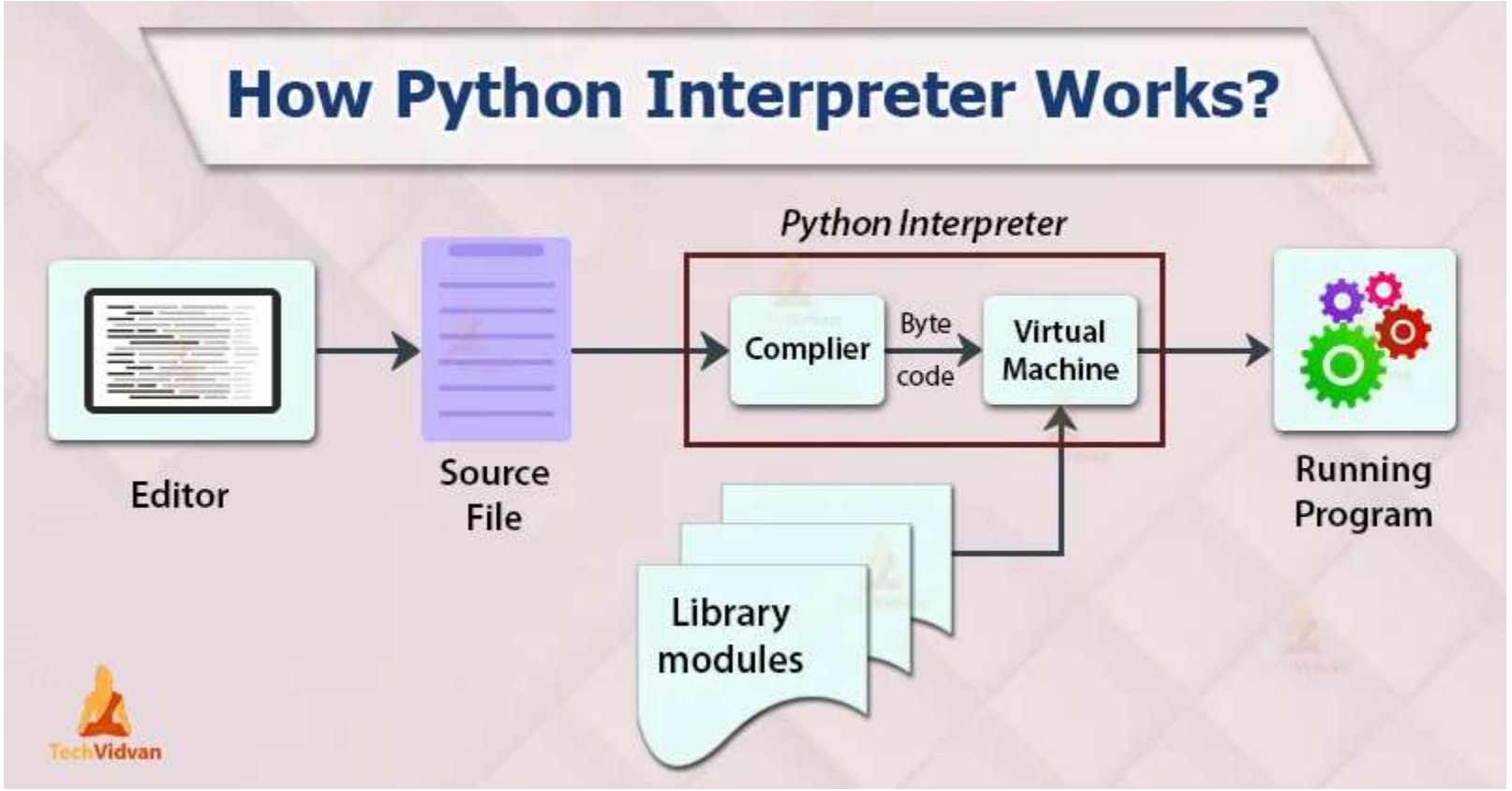
Compiler and Interpreter

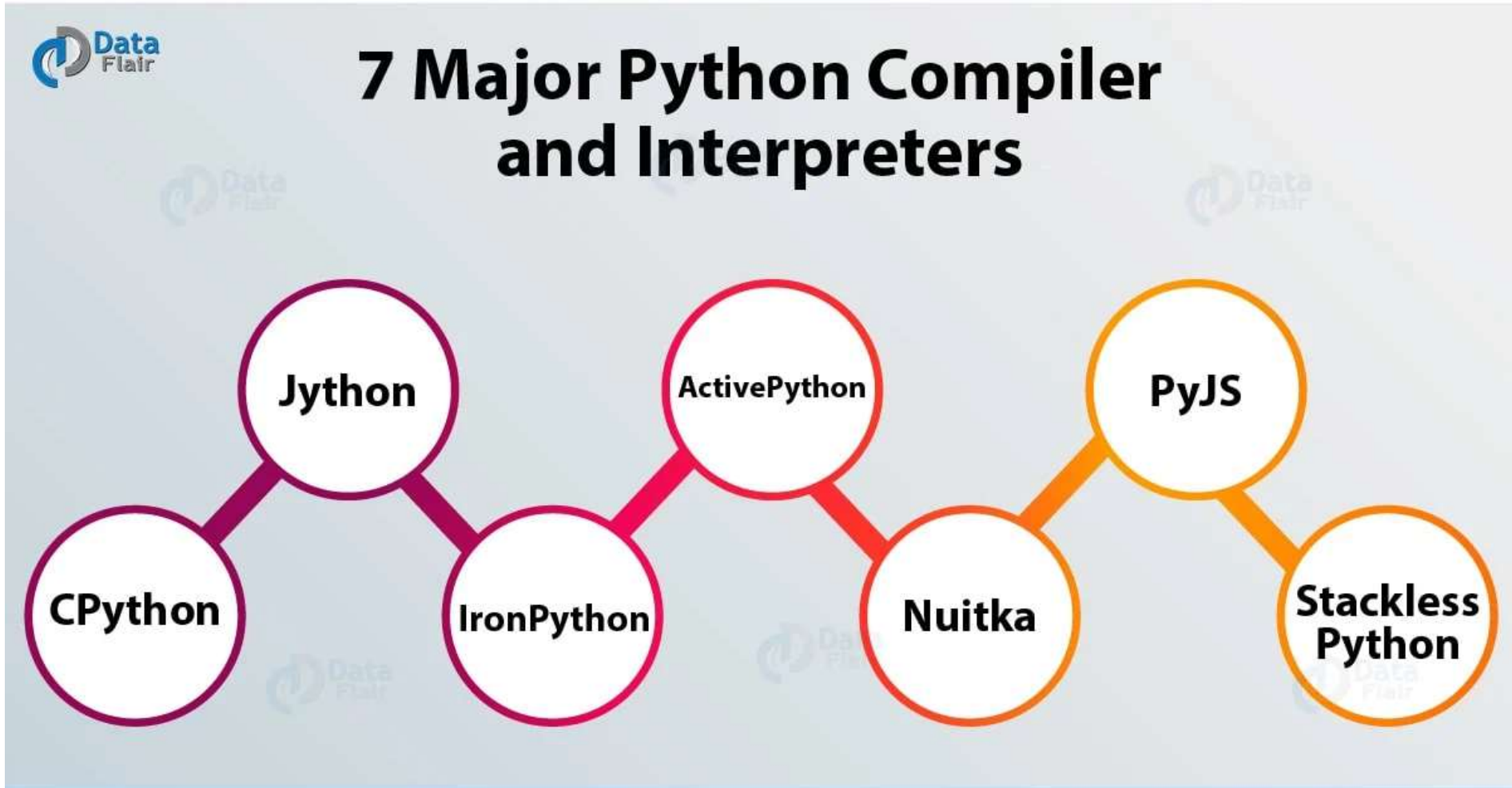
- A high-level language is one that is understandable by us, humans. This is called **source code**.
- However, a computer does not understand high-level language. It only understands the program written in **0**'s and **1**'s in binary, called the **machine code**.
- To convert source code into machine code, we use either a **compiler** or an **interpreter**.

Python Interpreters

Python is an interpreted language. Well, this is the way it goes.

- A compiler converts the .py source file into a .pyc bytecode for the Python virtual machine.
- A Python interpreter executes this bytecode on the virtual machine.




Data Flair
7 Major Python Compiler and Interpreters
Jython
CPython
IronPython
ActivePython
Nuitka
PyJS
Stackless Python

Python Compilers & Interpreters

- Cpython: This is the default and most widely-used interpreter written in **C.** This is an interpreter and has a foreign function interface with languages like C.
- Jython: Python interpreter written in **Java**. Jython takes Python code and compiles it to Java bytecode. This means we can run Python on any machine that runs a JVM (Java Virtual Machine). It supports static and dynamic compilation
- Iron Python: An interpreter with implementation around the **.NET Framework** and Mono. It supports dynamic compilation and an interactive console. Python scripts are capable of interacting with .NET objects.
- ActivePython: It is a Python distribution from ActiveState. It makes installation easy and cross-platform compatibility possible. Apart from the standard libraries, it has many different modules.

Python Compilers & Interpreters

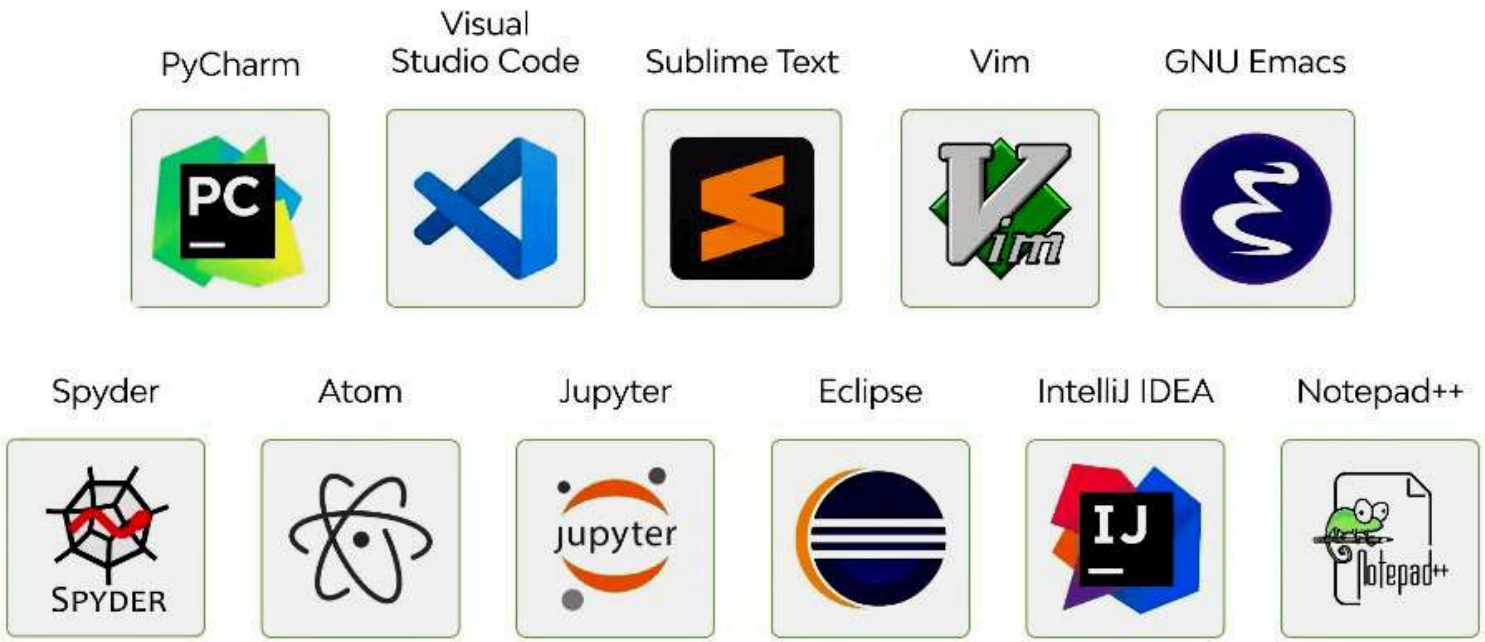
- Nuitka: Nuitka, source-to-source Python compilers that take Python code and ***compiles it to C/C++ executables*** or source code. Even when you don't run Python on your machine, you can create standalone programs with Nuitka.
- PyJS: PyJS is an internet application framework that will let you use Python to ***develop client-side web and desktop applications.*** You can run such an application in a web browser and also as a standalone desktop application. Earlier, it was called *Pyjamas*. It translates your Python code into JavaScript to let it run in a browser.
- Stackless Python: It is a Python interpreter. It is 'stackless' because it doesn't depend on the C call for its stack. It uses the C stack and clears it between calls. Stackless Python also supports threads and microthreads. Other than that, it provides tasklets, round-robin scheduling, serialization, and pre-compiled binaries.

Interpreter Vs Compiler

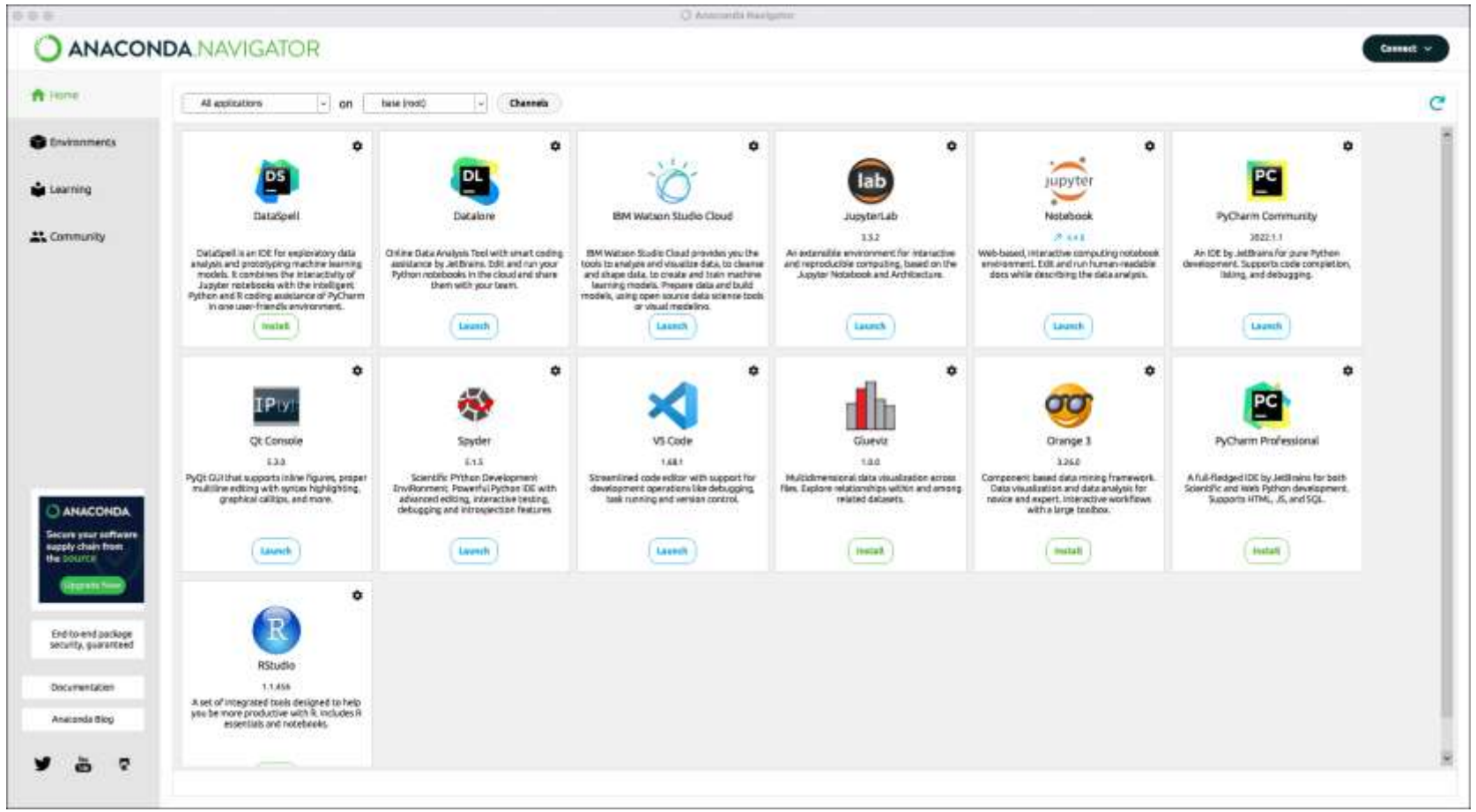
Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It never produces any intermediate machine code.	It generates an intermediate machine code.
Interpreters usually take less amount of time to analyze the source code. However, the overall execution time is comparatively slower than compilers.	Compilers usually take a large amount of time to analyze the source code. However, the overall execution time is comparatively faster than interpreters.
No Object Code is generated, hence are memory efficient.	Generates Object Code which further requires linking, hence requires more memory.
Programming languages like JavaScript, Python, Ruby use interpreters.	Programming languages like C, C++, Java use compilers.

Python IDE’s

Python IDE’s

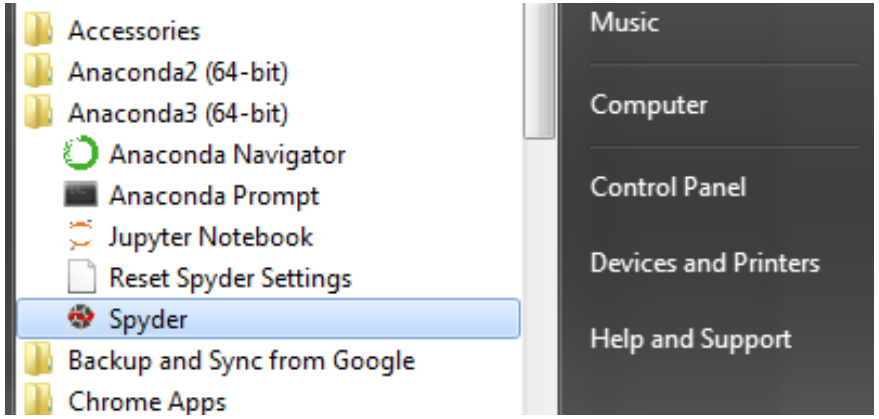

PyCharm
Visual Studio Code
Sublime Text
Vim
GNU Emacs
Spyder
Atom
Jupyter
Eclipse
IntelliJ IDEA
Notepad++
SPYDER
jupyter

The Anaconda .....

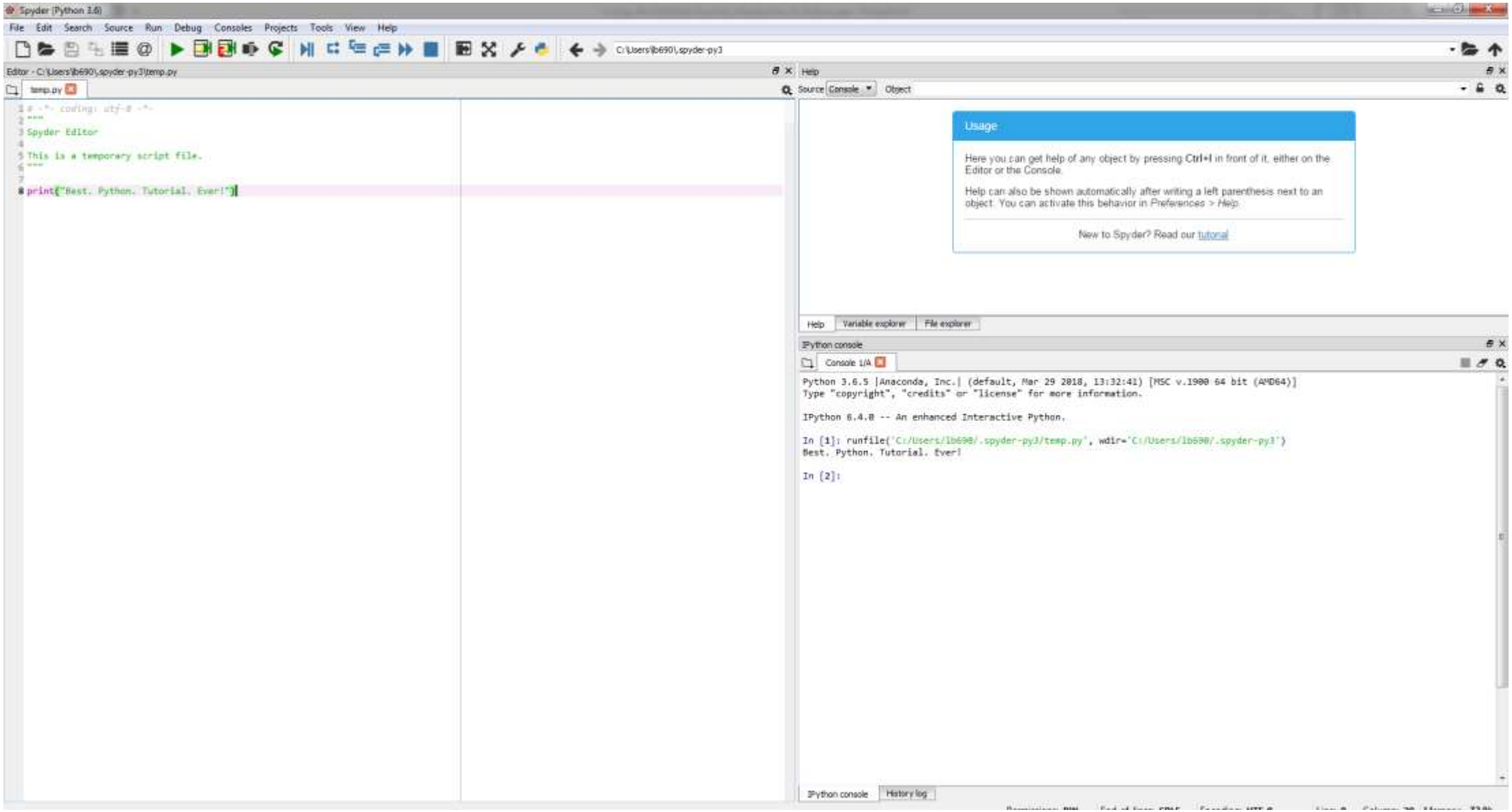


The Anaconda .....

- The Anaconda distribution is the most popular Python distribution out there.
- Most importable packages are pre-installed.
- Offers a nice GUI in the form of Spyder.
- Before we go any further, let's open Spyder:



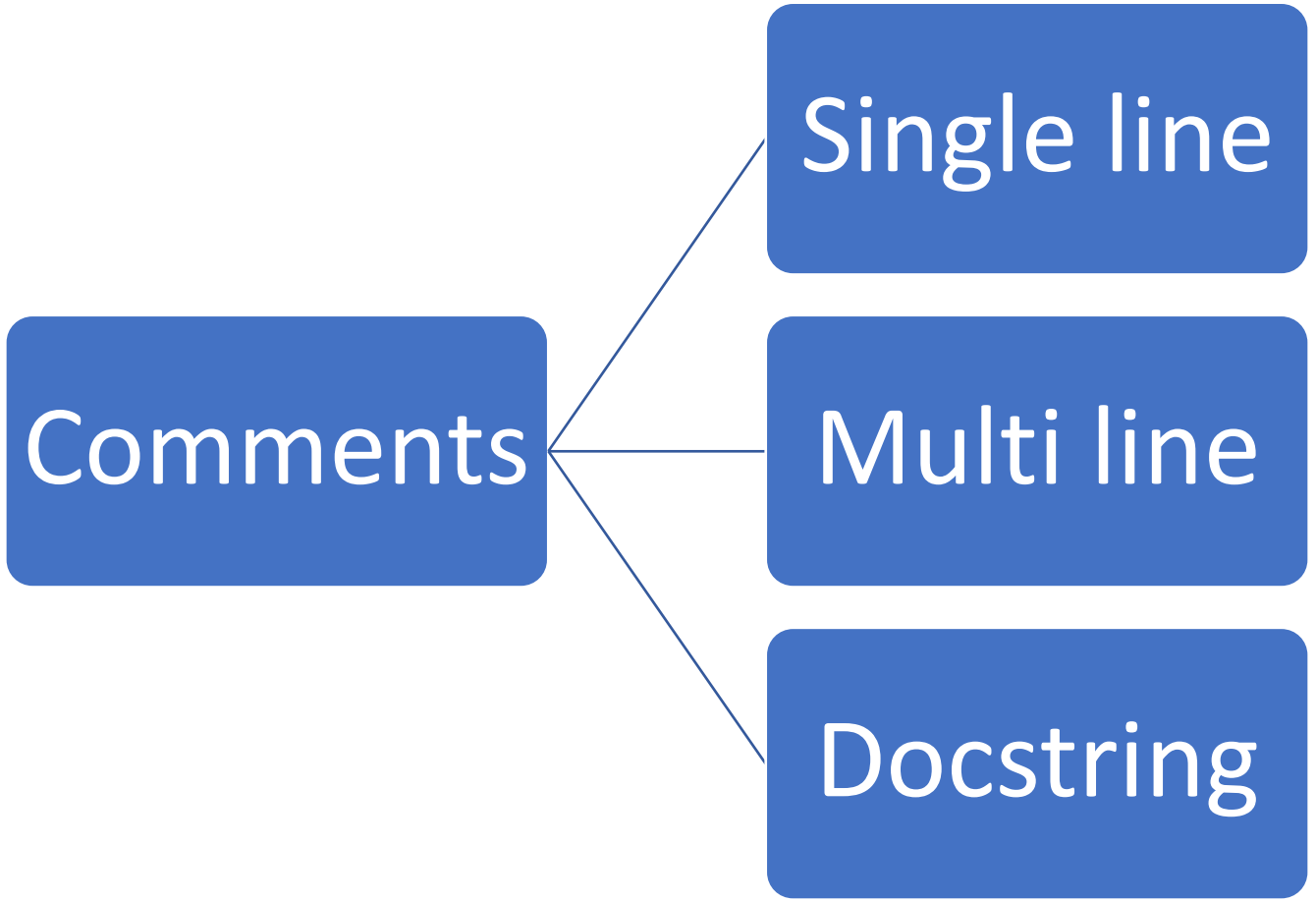
SPYDER


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\lb690\.spyder-py3
Editor - C:\Users\lb690\.spyder-py3\temp.py
temp.py
-*- coding: utf-8 -*-
"""
Spyder Editor
This is a temporary script file.
"""
print("Best. Python. Tutorial. Ever!")
Help
Source Console Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
New to Spyder? Read our tutorial
Help Variable explorer File explorer
IPython console
Console 1/A
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 6.4.0 -- An enhanced Interactive Python.
In [1]: runfile('C:/Users/lb690/.spyder-py3/temp.py', wdir='C:/Users/lb690/.spyder-py3')
Best. Python. Tutorial. Ever!
In [2]:
IPython console History log

Python Comments

- In Python, comments are used to provide explanations and notes within your code. Comments are not executed by the Python interpreter and are intended solely for human readers to understand the code better.

Python Comments



Single Line Comment

- **Single-line comments** are used to add explanations or notes on a single line of code. They start with the **#** symbol and continue until the end of the line.

In:

```
8 # This is a single-line comment
9 print("Hello, World!")  # This comment explains the purpose of the print statement
```

Out:

Hello, World!

Single Line Comment

- Python ignores the string literals that are not assigned to a variable so we can use these string literals as Python Comments.

In:

```
'This will be ignored by Python'
print("Hello, World!")
```

Out:

Hello, World!

Multi Line Comment

- Multi-line comments are often implemented using triple-quoted strings, which can be enclosed in either single (''') or double (""") quotation marks.

```
"""
This is a multi-line comment (docstring).
It can span multiple lines and is typically used to document functions or modules.
"""
```

```

'''
This is a multi-line comment (docstring).
It can span multiple lines and is typically used to document functions or modules.
'''
```

Docstring Comments

- Docstrings are used to associate documentation with Python ***modules, functions, classes, and methods.*** They serve as a way to describe what these code elements do and how to use them.

- Docstrings are ***placed immediately after the code element*** they are documenting, typically indented at the same level as the element's definition.

- Docstrings are enclosed in triple quotes, ***either single (''') or double (""") quotation marks.*** This allows for multi-line documentation.

- The content of docstrings is made available via the **__doc__** attribute or **help() function** associated with the code element. This makes it accessible for introspection and documentation generation tools.

Docstring Comments

In:

```
def multiply(a, b):
    """Multiplies the value of a and b"""
    return a*b


# Print the docstring of multiply function
print(multiply.__doc__)
```

Out:

```
Multiplies the value of a and b
```

In:

```
def multiply_numbers(a, b):
    """
    Multiplies two numbers and returns the result.

    Args:
        a (int): The first number.
        b (int): The second number.

    Returns:
        int: The product of a and b.
    """
    return a * b
# Access the docstring using help()
help(multiply_numbers)
print(multiply_numbers(3,5))
```

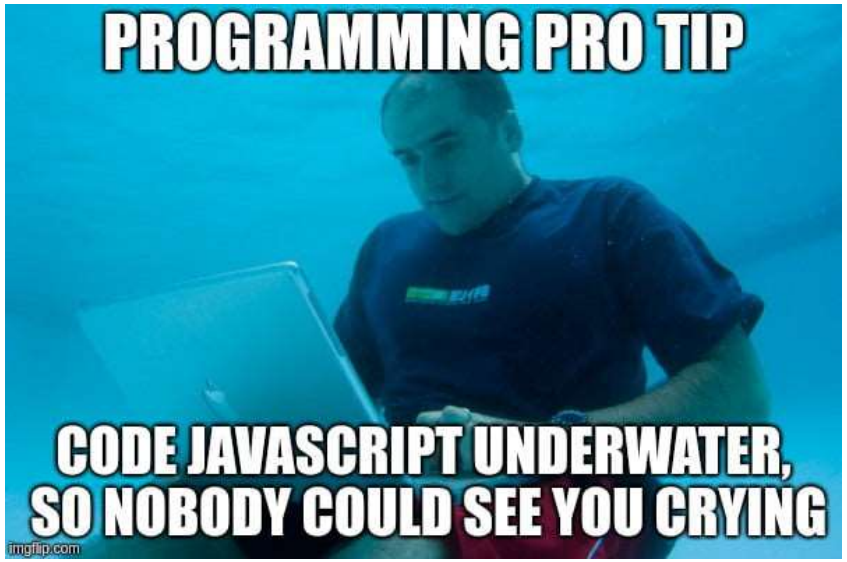
Out:

```
Help on function multiply_numbers in module __main__:

multiply_numbers(a, b)
    Multiplies two numbers and returns the result.

    Args:
            a (int): The first number.
            b (int): The second number.

    Returns:
            int: The product of a and b.

15
```

DEBUGGING

Debugging

- Debugging is in fundamental aspect of coding, and you will probably spend more time debugging than actually writing code.
- EVERYONE has to debug, it is nothing to be ashamed of.
- In fact, you should be particularly concerned if you do write a programme that does not display any obvious errors, as it likely means that you are just unaware of them.
- There are a number of debugging programmes available to coders. However, debugging the most common issues that you'll encounter when developing programmes can be done by following a few key principles.
- However, always remember that sometimes fixing a bug can create new bugs.



Print everything

- When debugging, the most important function at your disposal is the `print` command. Every coder uses this as a debugging tool, regardless of their amount of experience.
- You should have some sense as to what every line of code you have written does. If not, print those lines out. You will then be able to see how the values of variables are changing as the programme runs through.
- Even if you think you know what each line does, it is still recommended that you print out certain lines as often this can aid you in realising errors that you may have overlooked.

Print examples

Did this chunk of code run?

```
i = 0
while i < 6:
    string = '*'
    length = i * 3
    for j in range(length):
        string = string + '*'
    i += 1
print("Got here")
```

Yes, it did.

```
In [135]: runfile('//is
Got here
```

I want the value of variable to be 10 upon completion of the for loop. Did the for loop work correctly?

```
variable = 1
for i in range(10):
    variable += 1
print("variable", variable)
```

No.

```
In [133]: runfile('//
variable: 11
```

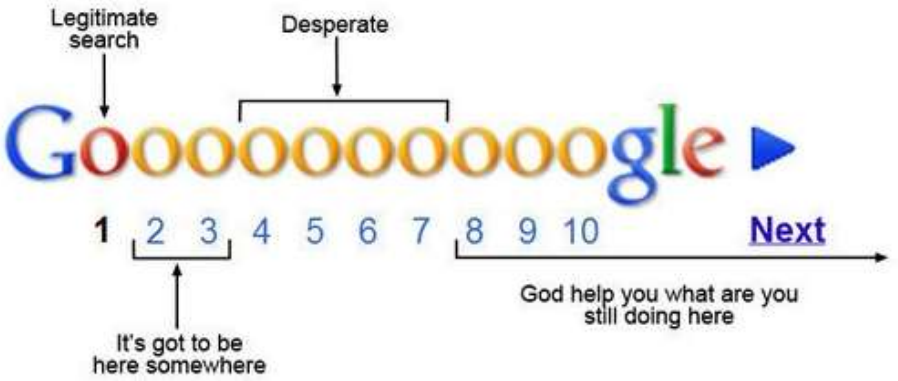
Run your code when you make changes

- Do not sit down and code for a hour or so without running the code you are writing. Chances are, you will never get to the bottom of all of the errors that your programme reports when it runs.
- Instead, ***you should run your script every few minutes***. It is not possible to run your code too many times.
- Remember, the more code you write or edit between test runs, the more places you are going to have to go back an investigate when your code hits an error.

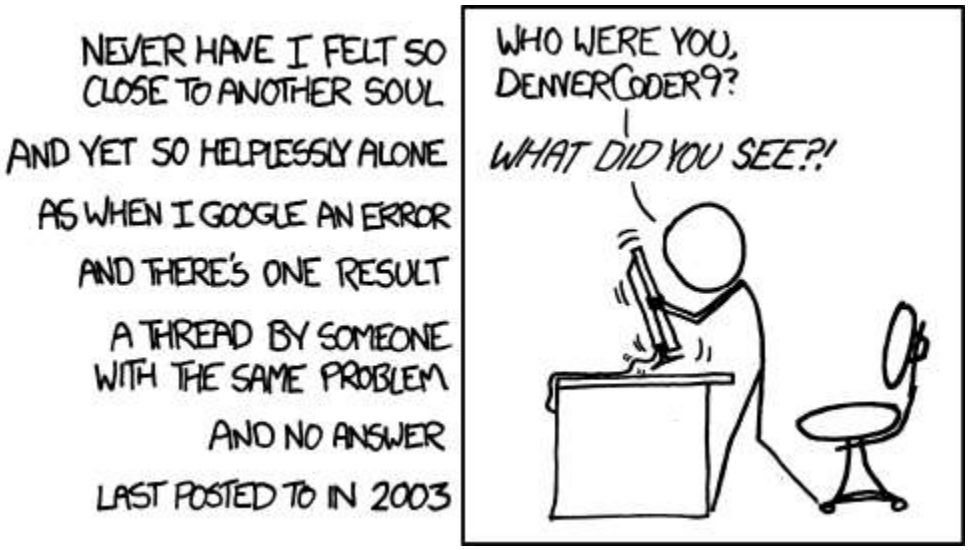
Read your error messages

- Do not be disheartened when you get an error message. More often than not, ***you'll realise what the error is as soon as you read the message;*** i.e. the for loop doesn't work on a list because the list is empty.
- This is particularly the case with Python, which provides you with error messages in 'clear English' compared to the cryptic messages given by offered by other languages.
- At the very least, the error message will let you know which lines is experiencing the error. However, this may not be the line causing the error. Still, this offers a good starting point for your bug search.

Google the error message



- If you cannot work out the cause of an error message, google the error code and description.
- This can sometimes be a bit of a hit-or-miss, depending on the nature of the error.
- If your error is fairly specific, then there will nearly always be a webpage where someone has already asked for help with an error that is either identical or very similar to the one you are experiencing; stackoverflow.com is the most common page you'll come across in this scenario.
- Do make sure that you read the description of the problem carefully to ensure that the problem is the same as the one you are dealing with. Then read the first two or three replies to see if page contains a workable solution.



Comment out code

- You can often comment out bits of code that are not related to the chunk of code that contains the error.
- This will obviously make the code run faster and might make it easier to isolate the error.

Binary searches

- This method draws upon a lot of the methods we have already covered.
- Here, you want to break the code into chunks; normally two chunks, hence this method's name.
- You then isolate which chunk of code the error is in.
- After which, you take the chunk of code in question, and divide that up, and work out which of these new chunks contains the error.
- So on until you've isolate the cause of the error.

Walk away

- If you have been trying to fix an error for a prolonged period of time, 30 minutes or so, get up and walk away from the screen and do something else for a while.
- Often the answer to your issue will present itself upon your return to the computer, as if by magic.

Phrase your problem as a question

- Many software developers have been trained to phrase their problem as a question.
- The idea here is that phrasing your issue in this manner often helps you to realise the cause of the problem.
- This often works!

Walter

Ask someone

- If all else fails, do not hesitate to ask a colleague or friend who is a coder and maybe familiar with the language for help.
- They may not even need to be a specialist, sometimes a fresh pair of eyes belonging to someone who is not invested in the project is more efficient at helping you work out your issue than spending hours trying to solve the issue on your own or getting lost the internet trying to find a solution.

Any Question So Far?