2020 Python for Absolute Beginners

Everything You Need to Program in Python

Learn Python programming from scratch with hands-on exercises in this beginner friendly Python book!

Robert Kean

TABLE OF CONTENS

What exactly is Python? Executive summary

Python is a high-level, object-oriented, dynamic-seed programming language. Combined with dynamic typing and dynamic referencing, its high-level data structures render it very appealing for Rapid Application Development as well as for use as a scripting or adhesive language for connecting established components. Python 's simple, easy-to - learn code information can make and

reduces the expense of maintaining the program. Python promotes modules and bundles that facilitate software modularity and reuse of code.
Python is a high-level, open-source programming language that acts as a general purpose language; it is most commonly compared to Ruby, JavaScript, and Scheme. What separates Python from other programming languages is that it is simple to use, can be taught to a novice, can be incorporated in any application, and can run on all modern operating systems, including Mac, Windows , and Linux; It is also one of the most powerful languages that a programmer can use, and is about three to five times faster than JavaScript and C++, respectively.

Relevant explanations why you need to use Python?.

You may use Python to build GUI mobile apps , websites and web applications. Additionally, Python, as a high-level programming language, helps you to concentrate on the main features of the code by taking care of specific programming tasks. The simple syntax rules of the programming language make it easier for you to keep the code base readable and the application maintained. There are also many explanations why you would favor Python over other programming languages.

1) Script Readable and Maintainable.

While writing a software application, you need to focus on the quality of your source code to make maintenance and updates easier. The Python syntax rules allow you to express concepts without writing additional code. Around the same time, Python, unlike other programming languages, stresses code readability and encourages you to use English keywords instead of pronunciations. You may also use Python to create custom apps without writing specific code. A functional and tidy code base can help you manage and upgrade the program without any additional time and effort.

2) Multiple programming paradigms.

Like other modern programming languages, Python also supports a number of programming paradigms. This completely promotes object driven and organized programming. The language features also support different principles of practical and aspect-oriented programming. Around the same

moment, Python often provides a complex form framework and automated memory management. The programming paradigms and language features allow you to use Python to build broad and complex software applications.

3) Compatible with global technologies and system

Python actually supports a variety of operating systems. You can even use Python interpreters to run your code on specific platforms and tools. Python is also an interactive programming language. This helps you to run the same app on different devices without recompiling this. So, after making any modifications, you are not forced to recompile the file. You can run the modified application code without recompiling it and check the impact of changes made to the code immediately. The functionality allows it easy for you to make improvements to the code without through the period of creation.

4)Simplify the development of complex software

Python is a general programming language of function. You will also use the programming language to build both desktop and mobile apps. You may also use Python to create advanced science and numerical applications. Python is developed with software intended to enable data processing and visualization. You may take advantage of Python's data processing capabilities to build personalized big data applications without any additional time and energy. Around the same time, Python's data analysis frameworks and APIs enable you interpret and view data in a more compelling and efficient way.

5) A range of open source platforms and resources

As an open source programming language, Python lets you substantially raising the expense of software creation. You may also use many open source Python modules, plugins and programming resources to reduce production time without growing development costs. You even have the option to choose from a wide range of open source Python frameworks and development tools based on your specific needs. For starters, you can simplify and speed up the creation of web applications utilizing robust Python web frameworks such as Django, Flask, Pyramid, Bottle and Cherrypy.

Installing and Running Python

You'll need to have access to the Python interpreter to get started working with Python 3. There are several common ways to do this:
Python can be downloaded on the Python Software Foundation website at python.org. Usually, this includes installing the correct software for your operating system and running it on your computer.
Some operating systems , particularly Linux, provide a package manager that can be run to install Python.
The best way to install Python 3 on MacOS is to install a package manager called Home brew. You will install applications that have a Python programming framework on smart phone operating systems including Android and iOS. It may be a perfect place to exercise your coding skills on the road.

Windows

It is extremely doubtful that the Windows machine has already been updated with Python. Normally, Windows computers don't. Fortunately, installation does not require anything other than uploading and running the Python update from the python.org website. Take a peek at how to run Python 3 on Windows:

Step 1 : Download the Installer for Python 3.

• Open a browser window and navigate to the Windows Download page on python.org.
• Below the caption at the top that says**Python Releases for Windows**, click the link for the**latest Python 3 release-Python 3.x.x.**(The newest in text is Python 3.6.5.)

Python 32-bit or 64-bit?
You may select either a 32-bit or a 64-bit installer for Windows. Here's what goes down to the disparity between the two:
If your processor has a 32-bit processor, you can use a 32-bit installer. On a 64-bit processor, any installer would probably function for most of the purposes. The 32-bit edition can typically consume less power, although the 64-bit edition is best optimized for heavy computing applications.
If you don't know which edition to pick, go for the 64-bit option.

Step 2: Execute the installer

After you have installed and downloaded the installer, simply run it by doubleclicking the downloaded link. There should be a dialog that looks like this:

Then just click the Install Now button. That's going to be what there is to it. You will have a functioning Python 3 installation on your machine a few minutes later. **OR**

Installing Python

• To run Python from the Windows Store:
• Go to the Start menu (lower left windows icon), type "Microsoft Store," click the button to access the shop.
• When the store is accessible, choose Check from the top-right toolbar, and type "Python." Select "Python 3.7" from the Applications list. Option to have.

• If Python has finished the download and installation phase, open Windows Power Shell using the Start menu (lower left windows icon). If Power Shell is open, enter Python — version to validate that Python3 has been enabled on your computer.

• Python's Microsoft Store installation includes pip, the standard package manager. Pip allows you to install and manage additional packages that are not part of the standard Python library. To check that you have usable pip for downloading and handling products, enter—version.

Install Visual Studio Code

By using VS Technology as your text editor / integrated programming environment (IDE), you can use IntelliSense (application completion aid), Linting (helps prevent error in your technology), Debug support (helps you identify errors in your application after you run it), Code snippets (small portable code block templates) and Unit checking (checking the user design for various input types)

VS Application often includes a built-in terminal that enables you to access a Python command line with a Windows Command prompt, PowerShell, or whatever you want, providing a smooth workflow between your code editor and the command line.

• Download Versus Update for Windows https:/code.visualstudio.com.

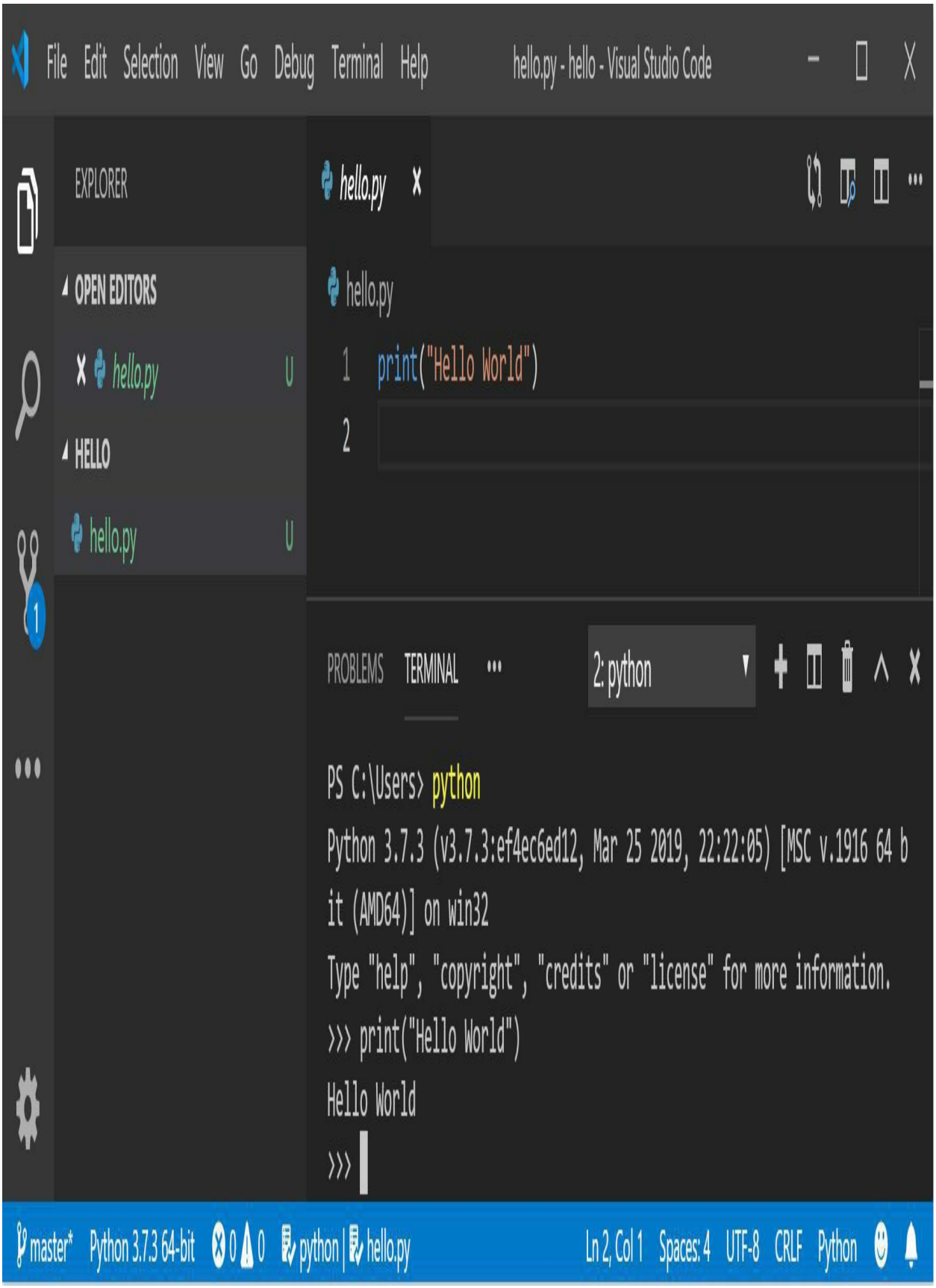
• If VS Code has been enabled, the Python extension must also be mounted. To install the Python version, pick the VS Package Marketplace connection or open the VS Application and check for Python in the Extensions menu (Ctrl+Shift+X).

• Python is an interpreted language, so if you want to run Python code, you need to inform the VS application which interpreter to use. We suggest sticking to Python 3.7 unless you have a particular justification to pick anything new. If the Python extension has been mounted, pick the Python 3 interpreter by opening the Command Palette (Ctrl+Shift+P), start typing the Python: Pick Interpreter command to scan, then select the script. You may also use the option Pick Python Environment on the bottom status bar if accessible (the chosen interpreter can already be shown). The command provides a set of accessible interpreters that VS Code may automatically locate, including virtual worlds.

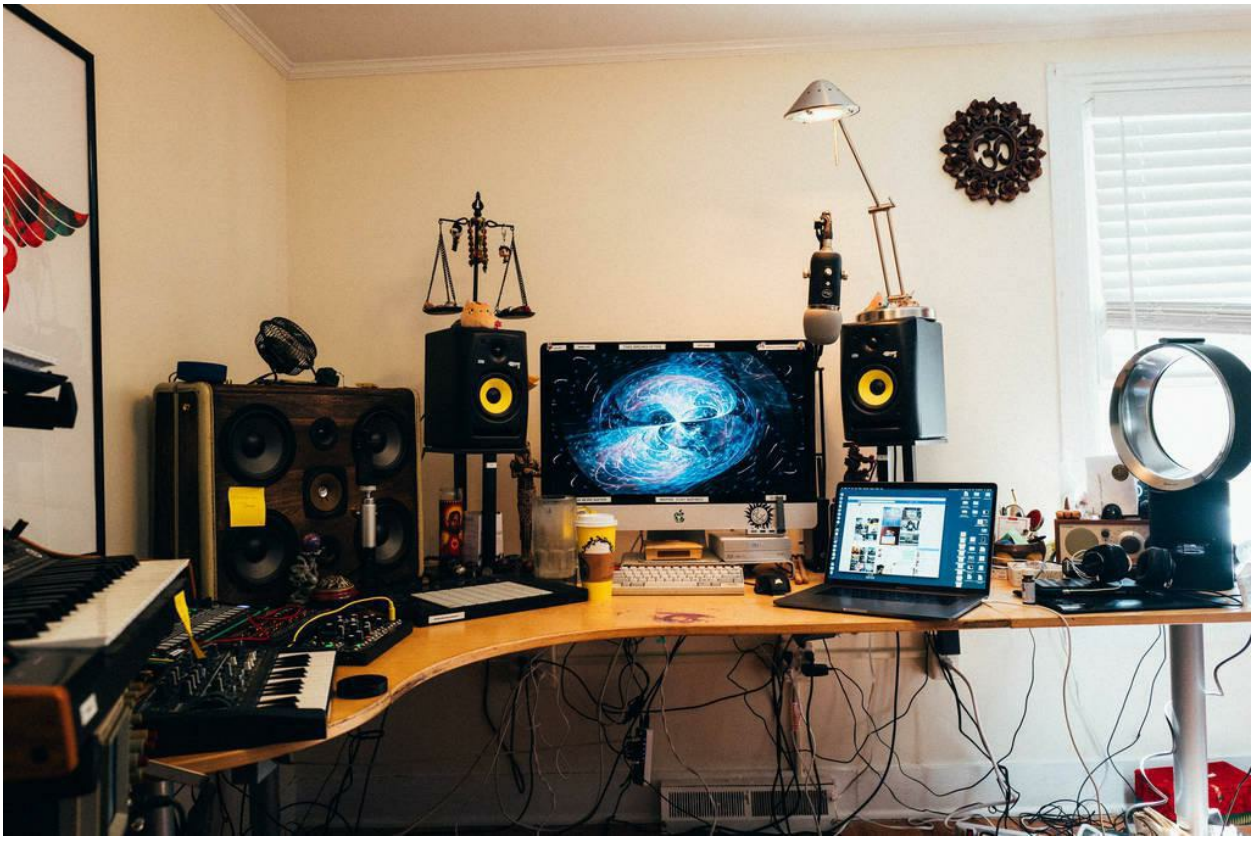
o To open the terminal in VS Text, pick View > Terminal, or alternatively using the Ctrl+' shortcut (using the backtick character). PowerShell is the main application.

o Open Python within your VS code terminal by simply entering the command: python

o Test the Python interpreter by entering: print("Hello World). Python will return your "Hello World" message.


File Edit Selection View Go Debug Terminal Help
hello.py - hello - Visual Studio Code
EXPLORER
hello.py
OPEN EDITORS
hello.py U
HELLO
hello.py U
hello.py
1 print("Hello World")
2
PROBLEMS TERMINAL
2: python
PS C:\Users> python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
master* Python 3.7.3 64-bit 0 0 python | hello.py
Ln 2, Col 1 Spaces: 4 UTF-8 CRLF Python

DEVELOPMENT ENVIRONMENTS



Write Python using IDLE or Python Shell is decent for basic tasks, but such frameworks easily transform bigger programming ventures into crushing pits of misery. Use an IDE, or even a decent dedicated application editor, allows coding easy.
What IDEs and Code Editors?
IDE (or Digital System Development) is a software development application. While the name suggests, IDEs incorporate a range of resources specially developed for software creation. Typically such methods include:

• Writer built to accommodate code (e.g. syntax highlighting and auto-completion) • Create, function, and test software
• Some type of control of the source

REQUIREMENT FOR A GOOD PYTHON CODING ENVIRONMENT

Store and reload your application files
When an IDE or editor doesn't require you to save your job and re-open it afterwards, it's not much of an IDE in the same condition as before you quit.

Run Code from within the Environment
Similarly, if you have to remove the Python code from the file, it's not anything more than a plain text editor.

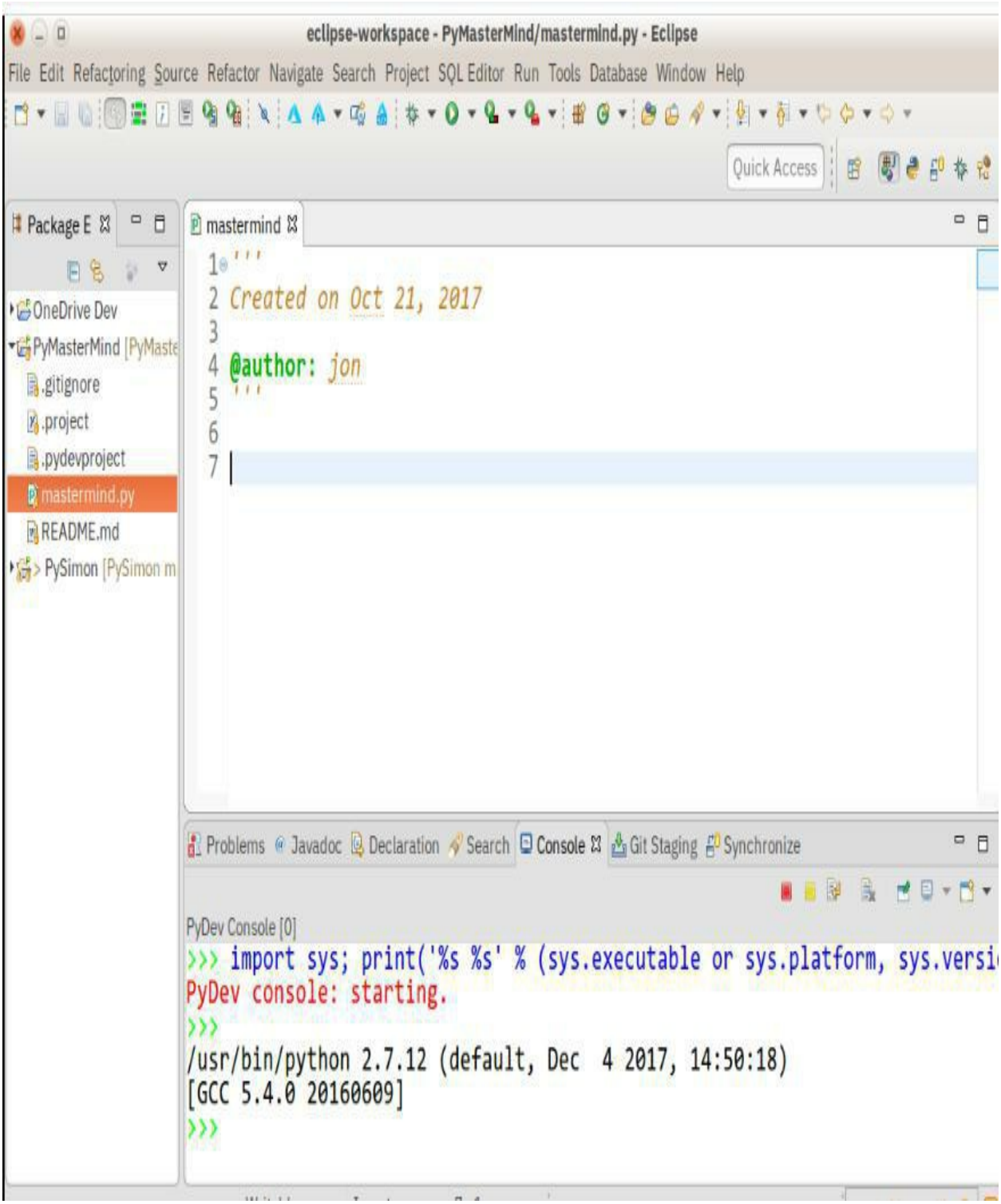
Debugging support
It's a central function in all IDEs and most decent application editors to be able to switch through the application when it runs.

Syntax highlighting
Being able to easily identify phrases, variables and icons in your application allows it far simpler to interpret and comprehend the application.

Quick printing of code
Any editor or IDE that is worth its salt should notice the colon at the end of a while or for a sentence and realize the next line will be indented.

Of course, there are a number of other options you might want, such as Source Code Control, Extension Template, Build and Test Software, Language Support and so on. Yet the above description is what I will consider as "key functionality" that a strong editing environment would help. With these features in mind, let's have a glimpse at some general purpose software that we can use to develop Python.

Eclipse or PyDev
Categories: IDE
Site: www.eclipse.org
PyDev tools, www.pydev.org
Eclipse is an open-source de-facto IDE for Java development. This has a vast ecosystem of plugins and add-ons, rendering Eclipse convenient for a wide variety of software practices.
Another of these enhancements is PyDev, which allows Python testing, application completion, and an immersive Python browser. Installing PyDev to Eclipse is easy: from Eclipse, select Help, Eclipse Marketplace, then check for PyDev. If required, press Download and restart Eclipse.



Pros: When you already have Eclipse mounted, it would be quicker and simpler to add PyDev. PyDev is very open to professional Eclipse developers.

Cons: Whether you're only starting off with Python, or with web creation in

general, Eclipse may be a lot to contend with. Remember when I said that IDEs are larger and require more knowledge to be used properly? Eclipse is all that and a (micro)chips bag.

WHAT IS A PROGRAM?

A program is a set of instructions that defines how to execute a computation. The Computing may be anything of a scientific sort, such as solving a series of equations or Finding the roots of a polynomial, but it may also be a mathematical operation, such as scanning and deleting text in a script, or something interactive, such as image processing, or Play the game. Definitions appear different in various languages, but there are a few simple guidelines in it. About every language:
Input: Get data from your computer, monitor, network, or other unit.
Output: Show the data on the computer, transfer it to the server, submit it across the network, etc.
Mathematics:perform basic mathematical operations such as addition and multiplication.
Conditional execution: Search for such requirements and use the correct code. **Repeat:**execute any activity frequently, typically with some variance

Compiler

A compiler is a computer program that transforms code written in a high-level programming language into a machine code. This is a software that converts the human-readable text into a language that the machine processor knows (binary 1 and 0 bits). The program reads the binary code in order to execute the relevant functions.

The programmer will conform with the syntax law of the programming language in which it is written. However, the programmer is just a tool and can not correct the errors contained in the software. Therefore, if you find a error, you ought to make adjustments to the program's syntax. Else, it's not going to list.

Interpreter

The interpreter is a computer program that encompasses any high-level system assertion in the machine code. This contains source code, pre-compiled code, and scripts. Both programmer and interpreter perform the same function that translates higher-level programming language to computer code. Nonetheless, before running the software, the compiler must translate the text to the computer code (create an exe). Interpreters translate text to computer code while the software is operating.

Basics of Python Programming

Variables, expressions and Statements.

One of the most important aspects of a programming language is the ability to modify it. Variable. A variable is a term which refers to a value.

Assignment Statements.

The assignment statement introduces a new attribute and assigns a name to it:
>>> message = ‘And now it’s completely different’
>>> n = 17
>>> pi =3.14111
This example is made up of three things. The first one assigns a string to a new variable named message , the second gives the integer 17 to n, the third assign the value of pi.

The standard way to describe variables on paper is to write a name with an arrow pointing to it.
For the interest of it. This sort of figure is called a state diagram since it indicates the condition of each system.

Variables

The Python variable is a fixed memory location for holding values. In other words, the variable in the python program transfers data to the machine for processing.

Each value in Python has a category of data. The various data forms in

Python are numbers, sets, tuples, sequences, dictionaries, etc. Variables may be identified by any designation or even alphabets such as a, aa, abc, etc.

How To Declare and use Varaible
Let's see the case. We must name the vector "a" and print it out.
a= 100 print(a)

Example 2.
# Declare a variable and initialize it
f = 0
print f
The re-declaration of the variable functions F = 'Hey sweet'
print f

Operators in Python are unique symbols that signify that any type of computation should be done. The principles on which the operator operates are called the**operands. Here is the Example**
>>> a = 10
>>> b = 20
>>> a+b
30
In this case, the + operator adds a and b to the operands together. The operand may be either a direct value or a variable that corresponds to an object:

Example 2.

>>> a =10 >>> b=20 >>> a+ b- 5 25

A sequence of operands and operators, such as + b-5, is called an expression. Python allows a variety of operators to transform data structures into statements. This are listed below.

Expression Statement
Expression statements are used (mostly interactively) to evaluate and write a number, or (usually) to call a procedure (a method that does not yield a significant result; in Python, procedures return the value None). Many types of speech statements are allowed and often helpful. The syntax for an speech

sentence is as follows:

Expression stmt: expression_list
An expression argument tests a set of expressions (which could be a single word). Through interactive mode, if the value is not Any, it is translated to a string using the built-in repr) (method and the corresponding string is written to the normal output (see section) on the line itself. (Expression statements containing None are not written, so that the call procedure does not trigger any output.)

Math Commands

The math module is a standard Python module and is often available. You need to load the software for this framework to use mathematical functions.
1. Ceil():-This function returns the smallest integral value larger than the number. When number is already integer, same number is retrieved.
2. Floor():-This method returns the largest integral value smaller than the total. Unless the sum is already integer, the same value would be retrieved.
 # import of "math" for mathematical operations Export mathematics a=2.3.

 # returning the ceil of 2.3
print ('The ceil of 2.3 is: ', end=') print the text (math.ceil(a))

 # Restoring the floor to 2.3 print ('The floor of 2.3 is: ', end=') print (math.floor(a))

Output:
Ceil of 2.3 is :3 The floor of 2.3: 2

3. Fab():- This function returns the absolute value of the number.
4. Factorial():-This method returns a number factorial. An error code would be shown if the number is not important.

 # Python application to reveal the workings of # fabs) (and fabs)
 # import of "math" for mathematical operations Import math

a=-10

b=5
 # The restoration of the absolute value. Print ('The absolute value of-10 is: ', end=') Print (math.fab(a))

 # the restoration of the 5 factorial Print ('The Factory of 5 is: ', end=') Printing (math.factorial(b))

Output:
The absolute value of-10 is 10.0. Factorial of 5 is: 120

List of Functions in Python Math Module.

• Ceil(x) Returns the smallest integer greater than or equal to x.
• Copysign(x, y) Returns x with the y symbol
• Fabs(x) Returns the total value of x
• Factorial(x) Recovers the same factorial
• Floor(x) Returns the largest integer less or equal to x
• Fmod(x, y) Returns the remainder if x is split by y
• Rexp(x) Returns the mantissa and exponent of x to the pair (m, e)
• Fsum(iterable) Returns the same number of the floating point values in the iterable
• Isfinite(x) Returns True if x is not an infinity or a -1 (not a number)
• Isinf(x) Returns Real if x is infinite positive or negative
• isan(x) Returns valid if x is a -1
• ldexp(x, I Returns * (2**i)
• Modf(x) Recovers all fractional and integer sections of x
• tronc(x) Recovers the truncated integer value of x
• Exp(x) Belongs to e**x
• expm1(x) Returns * * x-1
• Log(x, [basis]) Returns the logarithm of x to the origin (default to e)
• Log1p(x) Returns 1+x normal logarithm

• Log2(x) Recovers the logarithm of base-2 of x •
• Log10(x) Recovers the base 10 logarithm of x

• POW(x, y) Returns x lifted to Y strength • Sqrt(x) Recovers the same square root

Basic Data types

Data types are the classification or description of data objects. Data types are a category of value that decides which operations to be done on the information. Numeric, non-numeric and boolean (true / false) data are the most widely used data types. That programming language, though, has its own definition primarily representing its programming philosophy.

Python has the following normal or standardized data types

Numeric

A numeric value is any description of data that has a numeric meaning. Python defines
three kinds of numbers:

Integer : positive or negative integer numbers (without fractional part)
Float: Every real number with a floating point representation in which a fractional part is denoted by a decimal symbol or a mathematical notation.

Complex integer:an integer with a real and imaginary dimension defined as x+yj. x and y are floats and j is-1(square root-1 called an imaginary number)

Boolean's

Data with one of two built-in Truth or False meanings. Notice that 'T' and 'F' are stock. True and false booleans are not valid, so Python can create an error for them.

Sequence Types

A sequence is the ordered set of identical or similar types of data. Python has the following built-in data sequence types:

String: A string length is a set of one or more characters in single, double or triple quotations.
List: The list object is an organized set of one or more data objects, not always of the same kind, arranged in square brackets.
Tuple: A Tuple object is an organized set of one or more data objects, not always of the same kind, in brackets.

Dictionary
The dictionary object is an unordered data set in the context of a key: value pair. The list of such pairs is enclosed in curly brackets. For example: {1:"Steve, "2:"Bill," 3:"Ram, "4:" Farha”}

type() function
Python has a built-in type) (function to evaluate the data form of a certain attribute. For illustration, enter type(1234) in the Python shell and return <class 'int'>, which implies that 1234 is an integer value.

```
>>> type(1234)
<class 'int'>
>>> type(55.50)
<class 'float'>
>>> type(6+4j)
<class 'complex'>
>>> type("hello")
<class 'str'>
>>> type([1,2,3,4])
<class 'list'>
>>> type((1,2,3,4))
<class 'tuple'>
>>> type({1:"one", 2:"two", 3:"three"})
<class 'dict'>
```

Mutable and Immutable Objects
The data objects of the above forms are contained in the processing memory of the device. Many of such values can be updated through development, but the nature of others can not be altered until they are generated in the brain.

Number numbers, sequences, and tuples are eternal, which implies that their contents can not be modified after they have been formed.

At the other side, the set of objects in a list or dictionary can can be changed. It is easy to attach, remove, introduce and rearrange objects to a document or dictionary. We are also mutable objects.

Whitespace in Python

In Python3, string.whitespace is a pre-set variable used as a list constant. Throughout Python, the.whitespace string will send the characters space, window, linefeed, return, formfeed, and vertical window.

Syntax: syntax: string.whitespace
Parameters: Does not take any parameter since it is not a function.
Returns: Returns the character space, tab, linefeed, return, formfeed, and vertical row.
CODE 1.

```
# Import library string feature import string

Print ('Hello')
# Storing space for characters, tab, etc. Result = string.whitespace

# Printing a value Printing(result) Print ('heypretty')
```

OUTPUT
Hello
heypretty

Comments in Python

Comments are lines that appear in computer programs that are overlooked by compilers and translators. Including comments in programming allows the code more readable to humans because it gives some detail or clarification of what and section of the software is doing.

Depending on the function of the software, comments can act as instructions to you or warnings, or may be written with the goal of other programmers being able to appreciate what the code is doing.

Comment Syntax
Comments in Python begin with the hash (#) and whitespace

characters and proceed to the end of the line.
Generally, the comments should look like this:

this is a comment Since the comments don't work, you won't get any hint of the message when you run the software. Comments are in the source code for users to read, not for programs to perform.

Python Naming Rules and Conventions

- GENERAL

President of the Republic
Should not use terms that are too broad or too wordy. Find the best balance between the two.
Bad: data_structure, my_list, info_map, dictionary_for_purpose_of_storing_data_representing_word _definitions
Good: user profile, menu options, word definition
Don't be a jackass and call stuff like "O," "l" or "I"
By using the terms in CamelCase, capitalize both abbreviation letters (e.g. HTTPServer)

- PACKAGES

◆ Package labels will have a lower case
◆ If multiple terms are required, the underscore should distinguish them. ◆ It is typically best to adhere to 1 term name

- MODULES

◆ Names of the module will be all lower case
◆ If multiple terms are required, the underscore should distinguish them. ◆ It is typically best to adhere to 1 term name

- CLASSES

◆ Class titles would adopt the Upper Case rule.
◆ Python's built-in classes, though, are normally lowercase phrases. ◆ Exception classes will finish with "Bug"

-GLOBAL (MODULE-LEVEL) VARAIBLE

◆ Global variables will all be lowercase
◆ Words in a global variable name can be distinguished by an underscore ◆

- INSTANCE VARAIBLE

◆ The names of the variables of course should be all lower case
◆ Words in the name of the instance variable will be distinguished by an underscore ◆ Non-public instance variables will start with a single underscore
◆ If the name of the instance needs to be configured, two underscores can start the

name of the instance

-METHODS

◆ Method names will be mostly lower case terms
◆ Words in the name of the system should be distinguished by an underscore
◆ The non-public approach should start with a single underscore
◆ If the method name needs to be modified, two underscores can start the name of

the system.

- METHOD ARGUMENT

◆ The methods of course will have their first statement called 'self.' ◆ Class methods will have a first statement called 'cls'

- Functions

◆ Attributes of the function will be all lower case
◆ Words in the name of the function should be distinguished by an underscore

- CONSTANTS

◆ Constant terms deserve to be completely capitalized ◆ Terms can be distinguished by a constant name underscore

STARTING WITH CODE

-PYTHON HELLO WORD PROGRAM

Creating a simple program that shows any text is always the first thing every programmer does when setting up their software for the first time. It just requires a few seconds and will prove to the programmer that the system is set up and running. "Hello world" appears to be the most popular phrase to do while designing applications like this. I'm going to stick with the practice here, so feel free to compose anything you want!

However, you can implement a basic "Hello World" program by using Python's print() function to read the "Hello World" text to the computer.
Write down the program
Open the Python editor (IDLE is fine) and insert the following code:

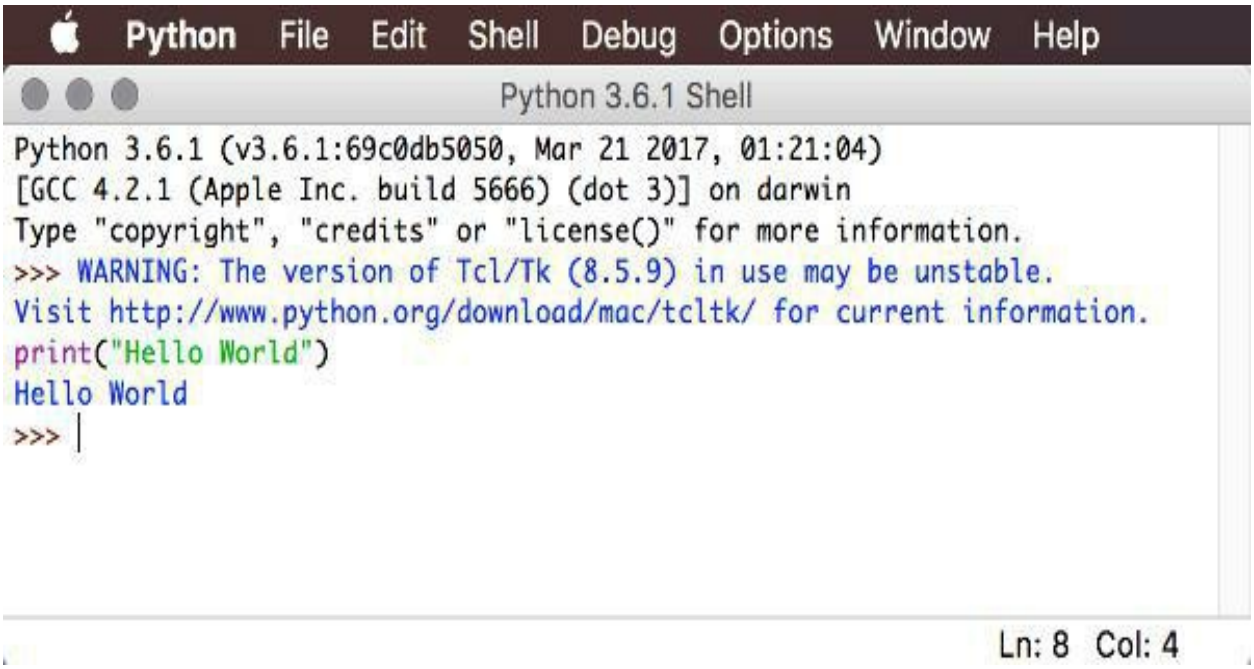
```
print("Hello World")
```

OUTPUT

```
RESULT

Hello World
```

This is definitely the easiest Python program you'll ever make, but it's still a Python program. Like for every computer program, you can save it to a register, and then add more features later if desired.



print('Hello World) 'in IDLE results in Hello World being written on the display.

SAVE YOUR PROGRAM

Go ahead and copy the template to a file named hello.py. This is normally achieved using Text > Save Editor or equivalent.

When saving your Python programs, using the.py file suffix. It is the suffix used for Python scripts.
The.py extension files can be accessed and modified with a text editor, but they need a Python interpreter to operate.

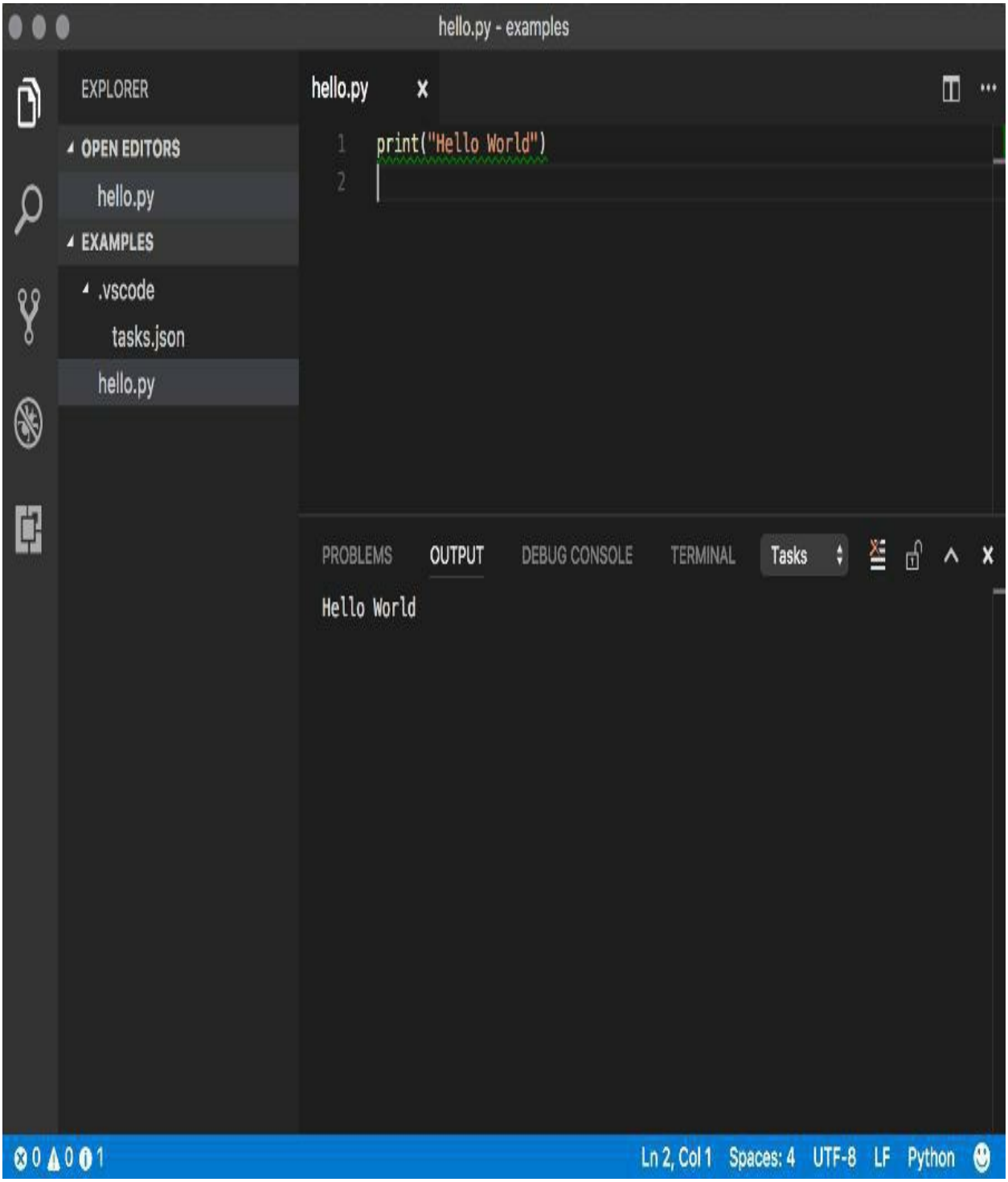
When you have a code editor designed for use with Python, you can find that this is a Python file and use the correct syntax for highlighting, debugging, etc.

RUN YOUR PROGRAM

Here is how to execute the program:

Enable the file (if not already opened). It is normally achieved using File > Open or similar.
Run the script right now. How to do this depends on your editor and your website. In IDLE, seek Run > Run Module (F5 shortcut).

- Using Ctrl+Shift+B on Windows and Linux or Cmd+Shift+B in Visual Studio Software



- PYTHON PROGRAM TO ADD TWO

NUMBERS

# Adds two numbers to this program Num1 = 1.5
Num2= 6.3

Add two numbers Sum = num1 + num2.
Display the number
Print{‘the sum of {0} and {1} is{2}’.format(num1,num2,sum)) Note. Here I used Python Online Compiler

MOST USED PYTHON OPERATORS

-Arithmetic Operators

Operator Name Operation + Addition a+b
- Subtraction a-b

* Multiplication a*b
/ Division a/b
// Floor division a//b

% Modulus a%b
** Exponential a**b

Result
Sum of a and b Difference of a and b
Product of a and b
Quotient of a and b
Floored quotient of a and b
Reminder of a and b
a to the power b

-Unary operations for + and

Operator Meaning Operation + Unary positive +a
- Unary negative -a

Result
a unchanged
a negated

-Comparison (Relational) operators

Operator Meaning Operation == Equal to a==b
!= Not equal to a!=b
> Greater than a>b
< Less than a<b
Result

True of a is exactly equal to b.
True if a is exactly not equal to b.
otherwise false
True if a is exactly greater than b. otherwise false
True if a is exactly less than b.
otherwise false

>+ Greater than or a>=b

equal to

<+ Less than or equal a<=b to
True if a is exactly greater than or
equal to b.
otherwise false
True if a is exactly less than or equal to b. otherwise false

-Logical Operators

Operator Operation and a and b
or a or b not not a==b

-Assignment Operators

Operator =
Operation c= a+b

+=
-=
*=

/=
%=

**= //=
a+=b a-=b
a*=b

a/=b
a%=b a**=b a//=b

Result
Assigns the value of its right operand/s to its left operand
Same as a= a+b

Same as a= a-b
Same as a= a*b

Same as a= a/b
Same as a= a%b
Same as a= a**b
Same as a= a//=b

-Bitwise Operators

Operator
|
^
&
<<
>>
Operation a|b
a^b
a&b
a<<b
a>>b
Result
True if both a and b are true. otherwise false
True if either a and b are true. otherwise false
If a is true then return false
Result
Bitwise or of a and b
Bitwise exclusive of a and b Bitwise exclusive of a and b A shifted left by n bits
A shifted right by n bits

-Ternary Conditional Operator

You can describe a conditional statement like this in Python:

```
x if C else y
```

The conditional (C) shall be evaluated first. When it returns False, the answer will be x, otherwise it will be y.

Example:

```
a = 7
print("Low" if a < 10 else "High")
```

RESULT

```
Low
```

PYTHON LISTS

A list is a set of attributes. You might see a list as a container that contains a set of objects in a specified order.
To build a list in Python, simply add any amount of comma-separated values within square brackets. Just like this:

```
planets = ["Earth", "Mars", "Saturn", "Jupiter"]
```

In Python, a list can contain objects of varying data types. For context, here's a list that includes a number, an integer, and a different set.
If we print both of these lists, we get the following:

RESULT

```
['Earth', 'Mars', 'Saturn', 'Jupiter']
['Hey', 123, ['Dog', 'Cat', 'Bird']]
```

-Accessing a List Item

Through referring to the index, you may pick a specific list object. Note, Python uses zero-based indexing, so continue counting at 0 while you do this. Now, to figure out what the second item in the planet list is, do this:

```
planets = ["Earth", "Mars", "Saturn", "Jupiter"]
print(planets[1])
```

You may also pick a set of objects by listing two indexes separated by a column. Just like this:

```
planets = ["Earth", "Mars", "Saturn", "Jupiter"]
print(planets[1:3])
```

RESULT

```
['Mars', 'Saturn']
```

Remember that this does not necessarily contain the upper index specified? I stated 1:3, but it returned only two values. It is since Python just returns the list items to, but does not include, the last listed table.

-Negative indexing

You should continue numbering from the end of the list using a negative index. In this scenario, the count must continue at-1 (not zero). So the index of-1 is the last item,-2 is the second item, and so on.

Example:

```
planets = ["Earth", "Mars", "Saturn", "Jupiter"]
print(planets[-2])
```

```
RESULT

Saturn
```

-Update a List Item

You can adjust a list item like this one:

```
planets[1] = "Mercury"
```

So it's like defining a variable, then applying the name of the element to the list object index in square brackets.
Here's a glimpse of:

- Append a List Item

You can add items to the list using the append()function. Below is an indication of this:

```
planets.append("Mercury")
```

Here's a glimpse of:

```
# Set the initial list
planets = ["Earth", "Mars", "Saturn", "Jupiter"]
# Print the initial list
print(planets)
# Add the new list item
planets.append("Mercury")
# Print the updated list
print(planets)
```

RESULT

```
['Earth', 'Mars', 'Saturn', 'Jupiter']
['Earth', 'Mars', 'Saturn', 'Jupiter', 'Mercury']
```

- Delete a List Item

Using the del keyword to delete a list object in a particular index. Just like this:

```
del planets[2]
```

Here's a fleeting glance of:

```
# Set the initial list
planets = ["Earth", "Mars", "Saturn", "Jupiter"]
# Print the initial list
print(planets)
# Delete the third list item
del planets[2]
# Print the updated list
print(planets)
```

RESULT

```
['Earth', 'Mars', 'Saturn', 'Jupiter']
['Earth', 'Mars', 'Jupiter']
```

PYTHON TUPLE

The Python tuple is identical to the list. The distinction between the two is that we can not modify the elements of a tuple until it has been allocated, while we can adjust the elements of a set.
A tuple is a set that is ordered and unchangeable. Tuples with round brackets are written in Python.
Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example

tup1 = ('Physics,' Biology, 1997, 2000);
tup2 = (1 , 2, 3, 4, 5) ;
tup3 = "a," "b," "c," "d"

The empty tuple is represented as two brackets containing nothing −

tup1 =().

You have to use a comma to compose a tuple containing a single number, even if there is just one number −

tup1 = (50,)
Like string indices, tuple indices start at 0, and can be cut, concatenated, and so on**.**

- Accessing Values in Tuples

To view the values in the tuple, using the square brackets to slic together with the index or indices to get the value present in that set. For examples, −

tup1 = ('Physics,' Biology, 1997, 2000); tup2 = 1 , 2, 3, 4 , 5 , 6, 7;
print 'tup1[0]:,' tup1[0];
print 'tup2[1:5]:,' tup2[1:5];

When the above code is executed, the following result is created −
tup1[0]: Physics: tup2[1:5]: [2, 3, 4, 5]:

- Updating Tuples

Tuples are immutable, which ensures that you can not edit or alter the values of the tuple components. You may take portions of current tuples to build new tuples as seen in the following illustration −

tup1 = (12, 34.56) tup2 = ('abc,' xyz);
The next action is not true for tuples **# tup1[0] = 100.**

Now let's make a fresh tuple as follows
tup3 = tup1 + tup2
Print up to 3;
When the above code is executed, the following result is created −

(12, 34.56, 'abc' and 'xyz')

- Delete Tuple Elements

The elimination of individual tuple components is not feasible. There is, of course, nothing inconsistent with bringing together another tuple with the unused components discarded.
Only use the del expression to directly delete the whole tuple. For examples,
− **tup = ('Physics,' Chemistry, 1997, 2000);**
print tup;
Del tup;
print "After removing the tup:;"
Print tup;
It resulted in the following result. Notice the exception made, that is because there is no longer a tuple after del tuple −

'Physics,' 'Chemistry,' 1997, 2000)
After deletion of the tup:
Traceback (last current call):
"test.py" file, line 9, in
print tup;
NameError: the name 'tup' is not specified (12, 34.56, 'abc' or 'xyz')

- Basic Tuples Operators

Tuples refer to + and * operators just like strings; they often imply concatenation and repeat, only that the product is a new tuple, not a series.
Python Expression Result
Len((1,2,3) 3

(1,2,3)+(4,5,6) (1,2,3,4,5,6) (‘Hi!,) (‘Hi!, ‘Hi!, ‘Hi!,‘Hi!,) 3 in (1,2,3) True

- Indexing , Slicing , Matrices Since tuples are sequences, indexing and slicing operate the same way for tuples as for strings. Assuming to follow data −

Description Length

Concatenation Repetition
Membership

L = ('spam,' Spam,' 'SPAM!')

Python Expression Result Description L[2] ‘SPAM!’ Offsets start at zero L[-2] ‘Spam’ Negative : count from the

right
L[1:] [‘Spam’ , ‘SPAM!’] Slicing fetching sections

-Tuple Methods

Python has two built-in methods that can be used on tuples.

1. (count) Returns the amount of times the defined value occurs in the tuple
2.**(index)**Checks the tuple for the defined value and returns the place where it was located.

PYTHON SETS

Mathematically, a set is a collection of objects that are not in any specific order. The Python package is identical to this abstract description with additional requirements below.
The components in the package can not be duplicated.
The components in the collection are immutable (can not be modified), but the system as a whole is mutable.There is no index attached to any part of the python package. They also will not endorse any indexing or slicing activities.

Grouping objects into a collection may also be useful in programming, and Python offers the built-in group class to do so. Sets are differentiated from certain categories of artifacts by special operations that can be done on them.

-Defining a Set

The built-in set type of Python has the following characteristics:

● Sets are not ordered.
● Set components are special to everyone. Duplicate components are not allowed. ● The set itself may be changed, but the elements contained in the

set will be of an

immutable form.

Python sets are usually used for mathematical operations such as union, intersection, difference and complement, etc. We may build a set, access its elements and execute such mathematical operations as seen below.

Creating a Set

The set is generated by using the collection) (method or by putting all the elements in a pair of curly braces.

Days = set(["Mon",""Tue",""Wed",""Thu",""Fri",""Sat,""Sun])
Months={"Jan",""Feb",""Mar}
Dates: {21,22,17}
print(Days)
print (Months)
print(Date)

When the above code is implemented, the following result will be produced. Please notice how the order of the elements in the test has shifted.

set(['Wed,' 'Sun,' 'Fri,' 'Tue,' 'Mon,' 'Thu' , 'Sat']) set(['Jan', 'Mar' , 'Feb'])
Set([17, 21, 22])

- Accessing Values in a Set

Within a set, we can not reach individual values. We can only view all of the components together, as seen above. So we can also get a collection of individual objects by looping around the package.

Days = set(["Mon",""Tue",""Wed",""Thu",""Fri",""Sat,""Sun])
For d in days: print(d)

When the above code is implemented, the following result will be produced.
Wed
Sun

Fri
Tue
Mon
Thu
Sat

- Adding items to a Set

We may attach elements to the collection using the ass() (process. Once again, as mentioned, there is no unique index attached to the newly added item.

Days = set(["Mon",""Tue",""Wed",""Thu",""Fri",""Sat])

Days.add('Sun)
print (Days)

When the above code is implemented, the following result will be produced.

set(['Wed,' 'Mar,' 'Fri,' 'Tue,' 'Mon,' 'Thu', 'Sat'])

-Removing Items from a Set

We may remove elements from a collection using the discard()process. Once again, as mentioned, there is no unique index attached to the newly added item.

Days = set(["Mon",""Tue",""Wed",""Thu",""Fri",""Sat])
Days.discard('Sun) print(Days)

When the above code is implemented, the following result will be produced.

Set(['Wed,' 'Fri,' 'Tue,' 'Mon,' 'Thu' , 'Sat'])

- Union of Sets

The union operation on two sets generates a new collection comprising all the distinct elements of both sets. In the illustration below, the "Wed" variable is present in both sets.

DaysA = set(["Mon",""Tue",""Wed])
DaysB = set(["Wed",""Thu",""Fri",""Sat",""Sun]) AllDays = DaysA

print(AllDay)

When the above code is implemented, the following result will be produced.
Set(['Wed,' 'Fri,' 'Tue,' 'Mon,' 'Thu', 'Sat'])

- Intersection of Sets

The intersection operation on two sets generates a new set comprising only the general elements of both sets. In the illustration below, the "Wed" variable is present in both sets.

DaysA = set(["Mon",""Tue",""Wed])
DaysB = set(["Wed",""Thu",""Fri",""Sat",""Sun]) AllDays = DaysA & DaysB
print(AllDay)

When the above code is implemented, the following result will be produced. Please notice that the outcome has just one "wed".
set(['Wed])

- Difference of Sets

The difference operation on two sets generates a new collection comprising only the elements of the first collection and zero of the second set. In the illustration below, the "We d" variable is present in both sets such that it can not be included in the result collection.

DaysA = set(["Mon",""Tue",""Wed])
DaysB = set(["Wed",""Thu",""Fri",""Sat",""Sun]) AllDays = DaysA – DaysB
print(AllDay)

When the above code is implemented, the following result will be produced.
Set(['Mon',' 'Tue'])

- Compare Sets

We will verify if a given set is a subset or a superset of another collection. The outcome is True or false based on the elements present in the set.

DaysA = set(["Mon",""Tue",""Wed])
DaysB = set(["Mon","Tue","Wed","Thu","Fri","Sat,"Sun]) SubsetRes = DaysA < = DaysB
SupersetRes = DaysB > = DaysA
printing(SubsetRes)
printing(SupersetRes)

When the above code is implemented, the following result will be produced.
True True

SET METHODS

● Add() Add an element to the set
● Clear() Removes all of the items from the set
● Copy() Returns a duplicate of a set Difference() Returns a set representing the difference between two or more sets
● Difference_update() Removes objects in this set that are also found in another set specified
● Discard() Delete the object you specified
● Intersection() Returns a set, i.e. the combination of two different sets.
● Intersection update() Removes objects in this set that are not present in any other specified set(s)
● Isdisjoint() Shows whether or not two sets have an intersection
● issubset() Returns whether or not any set includes this set
● Issuperset() Returns whether or not this set contains another set
● Pop() Removes an element from a set
● Remove() removes the the defined element
● Symmetric_difference() Returns a set of symmetrical variations between two sets.
● Symmetric_difference_update() inserts symmetrical variations between this set to a related one.
● Union() Returns the list comprising the set of union ● Update) update the set to the group of the sets and others

PYTHON DICTIONARIES

In the dictionary, each key is divided from its meaning by a column (:), the objects are split by commas, and the entire lot is enclosed in curly braces. An null dictionary without any objects is written with just two curly braces, like this: {}.

The keys are special to the dictionary, although the meanings might not be. The properties of the dictionary may be of any kind, but the keys must be of an eternal data form, such as sequences, numbers, or tuples.

- The dictionary is a list that is unordered, changeable and indexed. The Python dictionaries are composed with curly brackets and include keys and values.

Dict = {'Name': 'Zara,' 'Age': 7, 'Class': 'First'} print "dict['Name']:",dict['Name']
print "dict['Age']:", dict['Age']

When the above code is executed, the following result is obtained −
dict['Name]: Zara dict['Age]: 7
If we seek to enter a data item with a key that is not part of the dictionary, we get the following error −
dict = {'Name': 'Zara,' 'Age': 7, 'Class': 'First'} print "dict['Alice']: ", dict['Alice']
When the above code is executed, the following result is created −

dict['Alice]:
Traceback (last recent call): "test.py" file, line 4, in < module > Print "dict['Alice']:",dict['Alice'];" KeyError: 'Alice'

- Updating Dictionary

You may edit a dictionary by inserting a new entry or key-value pair, changing an existing entry, or removing an existing entry as seen in the basic illustration below.

dict = {'Name': 'Zara,' 'Age': 7, 'Class': 'First'} Dict['Age] '= 8; # modification of current entry dict['School] '= "DPS School;" # Apply new entry

Print "dict['Age']:", dict['Age'] Print "dict['School']:," "dict['School']
When the above code is executed, the following result is created −
dict['Age]: 8
dict['School]: DPS College

- Delete Dictionary Element

You may either delete specific dictionary components or clear all the contents of the dictionary. You may even delete the whole vocabulary with one process.
Only using the del statement to directly delete the whole vocabulary. Here's a easy example −

dict = {'Name': 'Zara,' 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove the entry with the word 'Name' dict.clear(); # remove all entries in Dict.
del dict; # remove the whole dictionary;
print "dict['Age']:", dict['Age']
print "dict['School']:", dict['School']

It resulted in the following. Remember that an exception is created as there is no longer a vocabulary after del dict −

dict['Age]:
Traceback (last recent call):
"test.py" file, line 8, in < module >
print "dict['Age']:", dict['Age'];"
TypeError: 'type' object is unsubscriptable

- Properties of Dictionary Keys

Dictionary standards do not have any limitations. They can be any random Python entity, either regular objects or user-defined objects. However, the

same is not true of the keys.
There are two essential things to note about keys to the dictionary −
(a) No more than one entry per key is permitted. That implies that no double key is permitted. If you find two keys during task, the last entry scores. For examples, −
dict = {'Name': 'Zara,' 'Age': 7, 'Name': 'Manni'} Print "dict['Name']:", dict['Name']
When the above code is executed, the following result is created −
dict['Name]: Manni

(b) The keys ought to be immutable. That implies that you can use sequences, numbers or tuples as dictionary keys, but anything like ['key'] is not permitted. Here's a easy example −

dic = {['Name']: 'Zara,' 'Age': 7} Print "dict['Name']:",dict['Name']

Once the above code is run, it should generate the following output.
Traceback (last recent call):
"test.py" file, line 3, in < module >
dic = {['Name']: 'Zara,' 'Age': 7};
TypeError: list objects are unhashable

- Dictionary Methods

● Clear() Extracts all the elements from the dictionary
● Copy() Returns a subset of the dictionary
● fromkeys() Returns a dictionary with the defined keys and value
● get() Returns the value of the identified key
● Items() Returns a list containing a tuple for main value pair
● Keys() Returns a list containing the keys to the dictionary
● Pop() Removes an element with the key defined
● Popitem() Eliminates the last key-value pair added

● Setdefault() Returns the value of the key you listed. If the key does not exist:
● insert the key with the value specified

● Update() Updates the dictionary to the required key-value pairs ● Values() Returns the list of all values in the dictionary.

PYTHON CONDITIONAL STATEMENT

The if statement is perhaps the most well-known concept in the computer programming environment. It helps you to make anything happen to your program only if the condition is valid. You should also determine what occurs if the statement is incorrect.

- IF Example

```
score = 56
if score > 50:
  print("Well done, you passed the exam!")
```

RESULT

```
Well done, you passed the exam!
```

In this case , the number 56 was allocated to the variable named score. We then used an if statement to verify if the value of the score variable is greater than 50. If that is the case, then we print out the message. When the score is smaller, we don't do something (i.e. no text is sent out).

- ELSE Example

You may also add another element of your if statement, which helps you of decide what occurs when the condition is incorrect. In other terms, if the situation is valid, you should make your program do one thing, and another if it isn't.

```
score = 43
if score > 50:
  print("Well done, you passed the exam!")
else:
    print("Haha... you FAILED!")
```

RESULT

```
Haha... you FAILED!
```

- ELIF Example

Moving a step forward, you can add one or more elif parts to the statement as well. The elif parts are essentially the "else if" part. In other words, they require you to add additional criteria that are evaluated only if the previous one (or elif) is incorrect.

Example 1 using Elif Line

```
score = 43
if score > 50:
  print("Well done, you passed the exam!")
elif score > 40:
    print("You failed... but heck... we'll hire you anyway!")
else:
    print("Haha... you FAILED!")
```

RESULT

```
You failed... but heck... we'll hire you anyway!
```

Example 2 Multiple Elif Line

```
score = 95
if score > 90:
  print("WTF... why on Earth do you wanna work here?")
elif score > 50:
  print("Well done, you passed the exam!")
elif score > 40:
    print("You failed... but heck... we'll hire you anyway!")
else:
    print("Haha... you FAILED!")
```

RESULT

```
WTF... why on Earth do you wanna work here?
```

- Indenting

Please notice the use of indenting. Indenting is an essential aspect of the syntax of Python. Python uses indenting to describe the classification of sentences.
Delete the indented line from the first illustration to see what happens:

```
score = 56
if score > 50:
print("Well done, you passed the exam!")
```

RESULT

```
IndentationError: expected an indented block
```

An IndentationError . The Python interpreter didn't like it.

Although you can get away with irregular indenting (for starters, unintentionally pressing twice), it's better to strive to maintain the indenting constant. It will aid a number in readability.

PYTHON LOOPS

- While Loop

The while loop is where you program a set of instructions to be executed continuously for as many times as a specified condition is valid.
Python has two types of loops; a while loop, and a for loop.

In a manner, while loops are sort of close to if statements, they just do anything if a certain hypothesis is true. But the distinction is, if the assumption is true, and if the argument must obey the instructions until the remainder of the system begins. A while loop, on the other side, repeats itself over and over again, until the state is incorrect. Furthermore, a while loop has to have something that varies every time it runs — otherwise it will continue indefinitely.

Though loops are sometimes used with a counter — increasing (or decrementing) amount. Just like this:

```
counter = 1
while (counter < 10):
    print (counter)
    counter = counter +1
```

```
RESULT

1
2
3
4
5
6
7
8
9
```

Here's what the program is going to do:

● Set the counter to 1
● Join a time loop that keeps repeated until the amount is less than 10. ● Print the counter value to the screen any time it repeats.
● Increase the counter by 1

The bit where the counter is incremented is a critical part of this process. If we didn't do so, the loop will carry on indefinitely. The program should have been trapped in an endless loop (not a positive thing).

- Breaking a Loop

You should use the break sentence to get out of the loop early.

- A Loop with Else

Python lets you add another part of your loops. It is equivalent to the usage of another if statement. This lets you know what to do when the loop condition is false.

```
counter = 1
while (counter < 10):
    print (counter)
    counter = counter +1
else:
    print("The loop has succesfully completed!")
```

RESULT

```
1
2
3
4
5
6
7
8
9
The loop has succesfully completed!
```

Remember that the remainder of the loop will not run if you interrupt the loop:

```
counter = 1
while (counter < 10):
    print (counter)
    counter = counter +1
    if counter == 5:
        break
else:
    print("The loop has succesfully completed!")
```

RESULT

```
1
2
3
4
```

- For Loop

The for loop is where you can iterate through a sequence (such as a number, tuple, dictionary, or string) or some other entity before you hit the last element in the set.

The for loop helps you to do items like this, execute an action against each object in the set. For eg, you might print out each object, check each object to another element, add each item to the list, etc.

The for loop is identical to the while loop in the way that you may make the software run a series of instructions repeatedly. The distinction is that the for loop iterates over a set, not against a predefined state.

Below is an example of this:
Here's what the program is going to do:

Build a list of planets that is named the planet.
Enter the loop that goes around each object in the list of planets. The I represents every object in the set. It should have been named something. For

eg, we should have named the world to help explain the stuff we 're going against.
Print the current object on the screen with each iteration.

- Breaking a Loop

You should use the break statement to get out of the loop early.

- A Loop with Else Python lets you add another part of your loops. It is equivalent to the usage of another if statement. It lets you know what to do when the loop is done.

```
planets = ["Earth", "Mars", "Neptune", "Venus", "Mercury", "Saturn",
"Jupiter", "Uranus"]
for i in planets:
    print(i)
else:
    print("That's all folks!")
```

RESULT

```
Earth
Mars
Neptune
Venus
Mercury
Saturn
Jupiter
Uranus
That's all folks!
```

PYTHON FUNCTIONS

Functions are a key part of Python (and most other programming languages). Python provides a vast range of built-in functions, however you may build your own functions as well.

- Defining a Function 1. Function blocks begin with the def keyword followed by the name of the feature and the parenthesis(( )).
2. Any input criteria or statements should be included inside these brackets. Under these parentheses, you may also specify parameters.
3. The first declaration of a function may be an optional statement-a text list of a feature or a form series.
4. The code block within each function begins with a colon (:) and is indented.

5. The [expression] return statement exits the function, and potentially returns the phrase to the caller. The return statement lacking any claims is the same as the return statement None

- Syntax

def functionname(parameters): "function_doctring"
Function_suite
return[expression]

- Functions that Print Staff

Of course, if you need to, you can create a print function. Only use the print() function within the program. Just like this:

Please remember two points about this:

We dropped the statement of return. We don't have to return the value since we're still printing it.
If we call this function, we don't need to use print() anymore. This is because the function is already performing that.

- Return Multiple Values

You may return several values from a function by extracting them from a comma. In addition, this produces a tuple that you can use as is or unpack to use the individual values.

Below is an example of a feature that returns many values You should tell the

parenthesis that the outcome is returned as a tuple.
You will unpack every of these values by attaching a square bracket function call containing the index value. Just like this:

```
# ...
# Call the function and print the result
print(basicArithmetic(3,30)[0])
print(basicArithmetic(3,30)[1])
print(basicArithmetic(3,30)[2])
print(basicArithmetic(3,30)[3])
```

RESULT

```
33
90
0.1
-27
```

PYTHON ARRAYS

An array is a series of objects located in adjacent memory positions. The intention is to place several items of the same kind together. It allows it easy to measure the position of each variable by merely applying the offset to the base number , i.e. the memory location of the first variable in the sequence (usually the name of the sequence).

For convenience, we may think of an array of stairs where a value is put on each move (let 's assume one of your friends). There, you can recognize the position of each of your friends by merely understanding the count of the move they 're taking. Throughout Python, the collection can be managed by a function called array.
We may be helpful because we simply have to modify a common data type attribute. Users may view lists as arrays. Nevertheless, the consumer can not constrain the form of objects contained in the catalog. When you build arrays using the array module, all the array components must be of the same kind.

-Creating an Array

```
# The Python Program to Display
# Generating an array
# importing "array" for the development of array Import array as array
# Generating an integer style sequence a = arr.array('i, '[1, 2, 3])
# Download the original collection
print ('The latest generated list is: ", end =") For i of the range (0, 3):
print (a[i], end =" ")
print()
# Build a float style list
b = arr.array('d, '[2.5, 3.2, 3.3])
# Download the original collection
print ('The latest generated list is: ", end =") For i of the range (0, 3):
Print (b[i], end =" ")
```

OUTPUT
The new array generated is: 1 2 3 The new array created is: 2.5 3.2 3.3

- Adding Element From the Array

The elements may be applied to the array using the built-in insert() function. Insert is used to insert one or more data elements in an array. On the basis of the criterion, a new dimension may be inserted at the beginning, end or any specified array index. Append() is often used to add the value mentioned in its arguments at the end of the array.

```
# The Python Program to Display
# Attaching the items to the array
# importing "array" for the formation of array Import array as arr
# array with int type
a = arr.array('i, '[1, 2, 3])
print ('Array before insertion: ', end =') For i of the range (0, 3):
print (a[i], end = ")
print()
# Insert array using
# to insert()function
Insert(1, 4)
```

```
print ('Array after insertion: ', end =') for i in (a):
print I end = '')
print) (Print)
# sort float array
b = arr.array('d, '[2.5, 3.2, 3.3])
Print ('Array before insertion: ', end =') For i of the range (0, 3):
print (b[i], end = '')
print()
# Remove an element using append() b. the append(4.4)
print ('Array after insertion: ', end ='' ') for i in (b):
print (i, end ='' '')
print()
```

OUTPUT

Array before insertion: 1 2 3
Array after insertion: 1 4 2 3
Array before insertion: 2.5 3.2 3.3 Array after insertion: 2.5 3.2 3.3 4.4

- Removing Elements from the Array

Elements may be deleted from the array by utilizing the built-in remove()function, but an error happens where an element does not appear in the set. Remove()approach only eliminates one element at a time, iterator is used to delete a number of components. Pop()function may also be used to delete and return an element from the list, but by default it removes only the last element of the array, to extract an element from a different array location, the element index is provided as an argument to the pop()method.

```
# The Python Program to Display
# Removal of the elements in the array # to import "array" for array operations Import the array
# initializing an list of array values
# initializes an array of signed integers array = list.array('i, '[1, 2, 3, 1, 5])
# Print the original array
```

Print ('The latest generated list is: ', end =') For i of the size (0 , 5):
Print (arr[i], end =" ")
Print ("\r")

#using pop() to remove element at 2nd function Print ('The pop-up dimension is: ', end ="") Printing (arr.pop(2))
Print the array after popping

Print ('The list after popping is: " , " end =') For i of the range (0, 4):
Print (arr[i], end = " ")
Print("\r")
# Use remove()to delete 1st instance of 1 Arr.remove(1)
# Print array upon delete
Print ('The array after elimination is: ", end ="") For i of the range (0, 3):
Print (arr[i], end = " ")

OUTPUT
The new current array is : 1 2 3 1 5 The poped element is : 3
The array after popping : 1 2 1 5 The array after removing : 2 1 5

PYTHON CLASSES AND OBJECTS(OOP)

- Object Oriented Programming

Python is a normally consume programming language. This embraces growing programming approaches.
One of the common ways to solve a programming problem is by constructing objects. It is related to as Object-Oriented Programming (OOP).

There are two characteristics of an object:
● Characteristics ● Behaviour

- Class

The class is the object model.

We may think of class as a drawing of a parrot with a name. This includes all

the specifics of the label, colors, scale, etc. In the basis of such definitions, we will research the parrot. A parrot is an entity here.

The definition for the parrot class may be

```
Class parrot:
pass
```

Here, we use the keyword class to describe the empty Parrot class. We construct instances from class. An instance is a particular entity generated by a class.

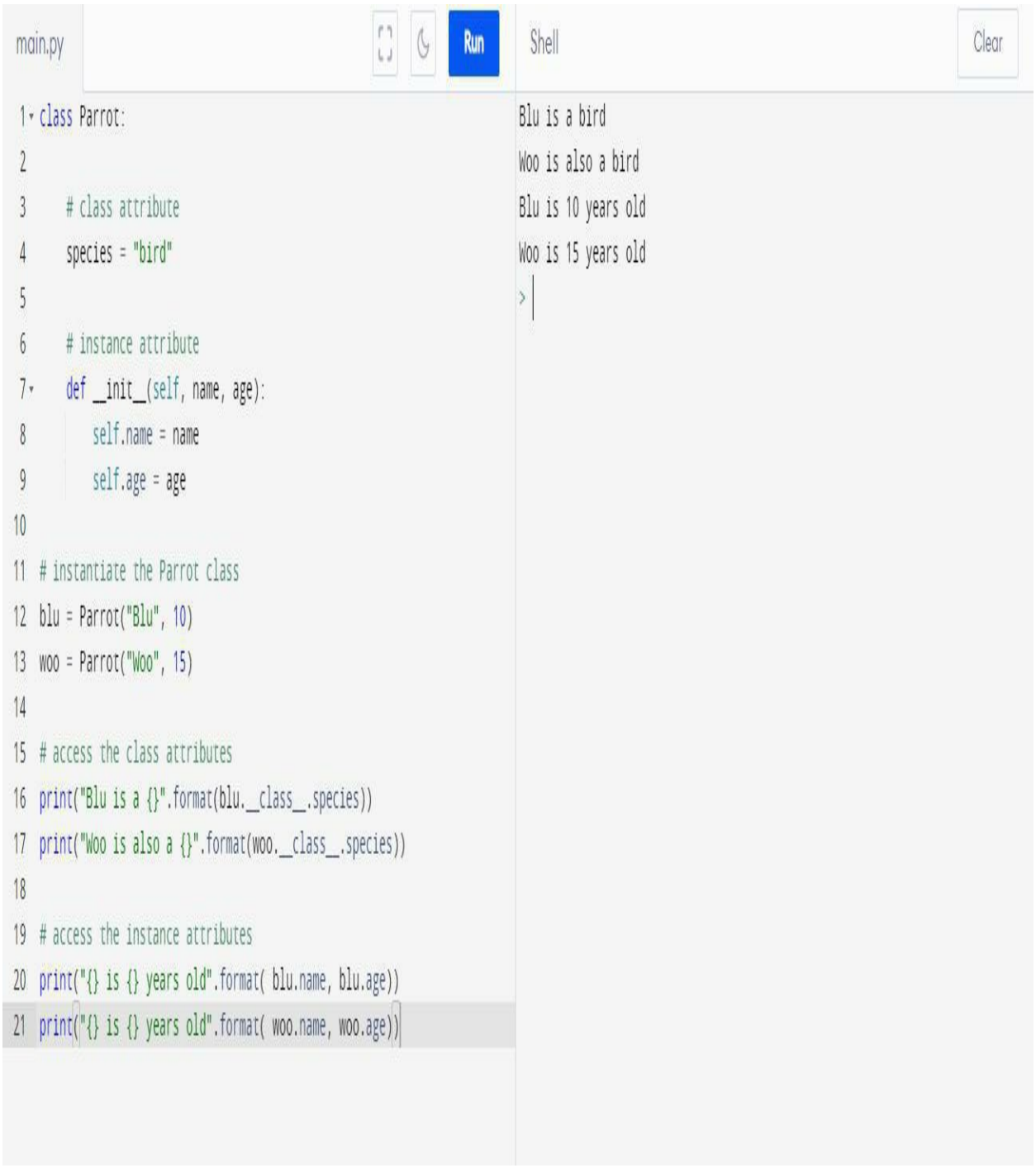
- Object

An object (instance) is a class snapshot. As the class is formed, only the definition of the object is described. There is also no capacity or data reserved.

The explanation for the parrot class object may be: obj = parrot()
Here, obj is the object of the Parrot class.

Suppose we've got the description of parrots. Now, we're going to see how to create the parrots' class and Objects.

Example 1: Creating Class and Object in Python



We built a class with the name Parrot in the above system. Then, we 're describing qualities. The attributes are a function of the object.
Such attributes are specified by the class__init__ method. This is the initializer method that will be started as soon as the object is formed.
Then, we build the Parrot class instances. Here, blu and woo are references (value) to our new objects.

We may control the class attribute by using _class_.species. Class features are the same for all the feature cases. Similarly, we will use blu.name and blu.age to control the instance attributes. Nevertheless, the instance characteristics are specific for every instance of a class.

- Encapsulation

Using OOP in Python, we can limit access to methods and variables. This prohibits direct alteration of data called encapsulation. Throughout Python, we designate private attributes that use underscore as a prefix i.e. _single__ double .

- Polymorphism

Polymorphism is the capacity (in OOP) to use a can multi-form interface (data types). Suppose we choose to colour the form, there are a variety of different design choices (rectangle, line, circle). However, we may use the same approach for coloring every form. This definition is known as polymorphism.

-Key points to Remember

● Object-oriented programming allows the software both simple to grasp and efficient.
● So the class can be exchanged, the code can be reused.
● Data is safe and accessible through code abstraction.
● Polymorphism requires the same interface to be used by different objects, meaning that programmers can create productive code.

PYTHON GLOSSARY

- Variables

For Python, variables are a reference for text and numbers.

- Interpreter

The Python interpreter is normally mounted on computers in / usr / local / bin / python on machines Where that was accessible.

- Comment

While interacting in every programming language, you make notes about the work in the code.

- Booleans

Boolean values are the two permanent False and True objects.

- Regular Expressions

Regular Expressions is a string template written in a concise syntax that allows one to easily test if a given string fits or includes a specific pattern.

- Math

The release of Python requires the Python interpreter, a rather basic creation. Setting, referred to as IDLE, databases, tools, and documents.

-Strings

A string is an order sequence of characters.

-List

The easiest data structure in Python is used to store a list of values.

- Dictionary

Dictionaries are sets of "key" and "value" objects.

- Loops

They need to repeat, loop back through the same block of code again and again to maintain a code performing valuable work.

- Function

A function is what you might name (possibly with certain conditions, objects you place in the brackets) that performs an operation and returns a result.

- Modules

Python modules allow programming a lot simpler. It's essentially a file that consists of a document that's already written.

CONCLUSION

Python is a high-level, structured and general-purpose functional programming language that emphasizes on readability of text. Python syntax helps programmers code in less phases relative to Java or C++.
Python has diversified uses in software technology companies such as games, online platforms and apps, language creation, prototyping, visual design uses, etc. It makes the language a greater abundance than many programming languages employed in the business.
Python is a versatile programming language that makes efficient usage of code lines, installation can be performed in a fantastic way, and debugging can be achieved easily. This has been particularly relevant around the globe because Google company has declared it one of the main programming languages.
We've seen a user-friendly and concise solution to Python. Python is the youngest of all the languages but it also gives a lot of frameworks. Python is a strong language all over, but it only excels with the ability to bring a lot of features with only a little bit of coding. Python is also preferred because of its usability and the guarantee that Python can stay open source forever.