Python Basics

Learn Python Easy Way

Comments on ‘Python Basics - Learn Python Easy Way in 1 Hour | Python For Cloud EP1’

Remember to keep comments respectful and to follow our Community Guidelines

Highlighted comment

[redacted] 6 hours ago

so good yaar, it is very helpful for beginners like me, lots of thanks and congrats, please continue

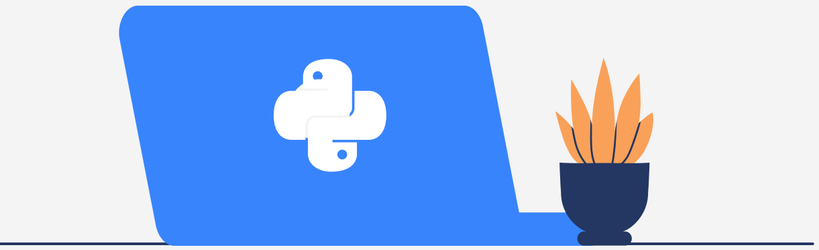
Python For Cloud EP 1

python

Bacis

Topics

- What is Python?
- Python History
- Python Use cases
- Installing Python
- First Program
- Python Keywords
- Python Variables
- Python Data Types
- Data Type Conversion
- Python Operators
- If..else
- For Loop
- While Loop
- Functions
- Modules
- Error Handling



What is Python?

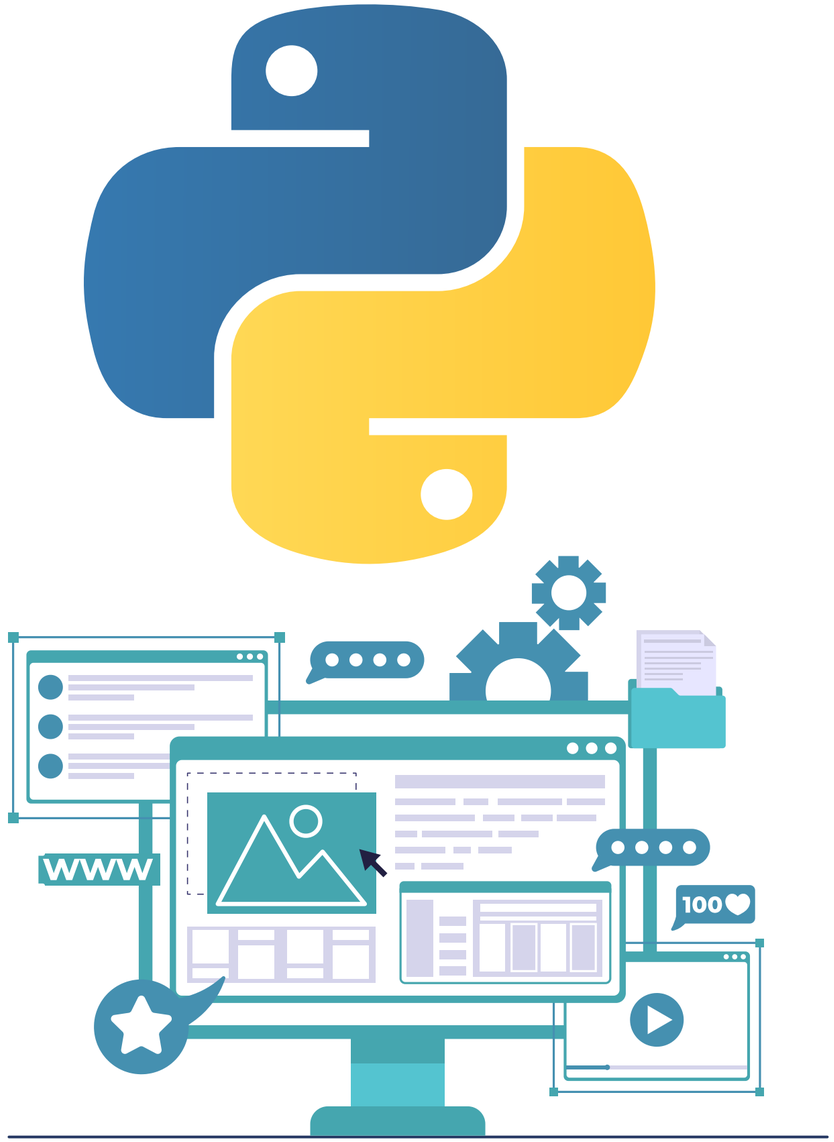
Python is an interpreted high-level general-purpose programming language. It consistently ranks as one of the most popular programming languages.

Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small, mid, and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented, and functional programming.

Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

Source: Wikipedia



Python Jourey

It took a long ...

It's all started by Guido van Rossum

1980

First Released as Python 0.9.0

1991

Python 2.0 was released

2000

Python 3.0 was released

2008

Source: Wikipedia

Use Cases

Where Python getting used the most?

Application Developmet

Python gets used for developing web applications, REST APIs, etc

Using: Django, Pyramid, Bottle, Tornado, Flask, web2py

Automation

Python gets used to developing workflow automation scripts including Cloud automation & resource provisioning.
Notably: Ansible, Salt, Openstack, xonsh written in Python

Data Science & Machine Learning

One of the most popular use cases of Python is in the field of Data Science and Machine Learning

Using: SciPy, Pandas, IPython, NumPy etc

Our Goal

Our goal in this series will be to have enough Python knowledge to work on Cloud & DevOps Projects, use different SDKs provided by the Cloud Service providers and other open-source projects

It's time to Install Python

01 Go to Python Official Web Site

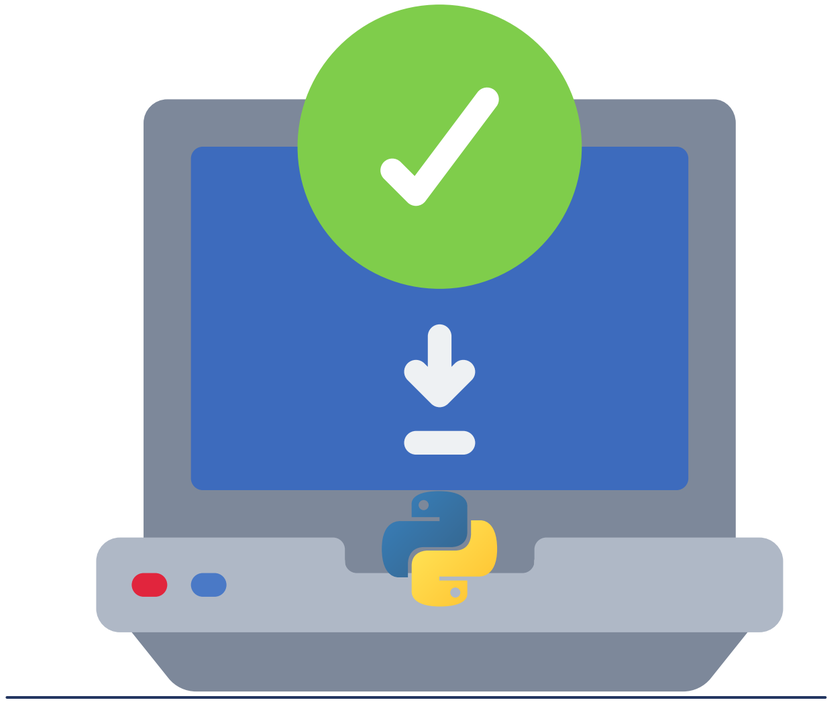
The first step is really easy and quick, visit https://www.python.org

02 Click on Downloads Menu

Click on the Download top menu and select for which Operating System(OS) you want to install Python

03 Follow rest of process as Instructed

For Mac and Windows, simply download, install and run! But for Linux and other OS further steps are needed as mentioned on that page. After installation is done, check the Python version by running: python --version



Getting Started...

01 Make Sure You have good An integrated development environment (IDE)!

Good IDE makes life easier, if you are not sure which to choose, here are the most popular for Python:
VS Code, PyCharm, Atom, Sublime, Jupyter
My personal all-time favorite is VS Code, which you can download from here: https://code.visualstudio.com

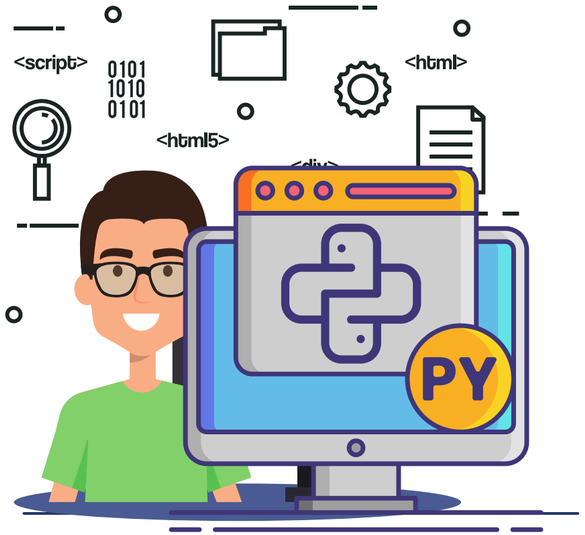
02 IDE installed ? let's run our sweet old first program!

hello_world.py

```
print("Hello, World!")
```

Note

In other programmig laguages, such as c , C++ , JavaScript, Java uses {} to define a block of code, however in Python uses **indentation**



Key Words

Python has reserved keywords

Python has multiple reserved keywords which we can not use as variable names or any identifier and these keywords have specific purposes/use cases while doing programming with Python (We will cover in the next sections)

List of keywords

if, elif, else, and, or, not, with, for, global, import, async, await, None, pass, return, raise, except, lambda, try, while, nonlocal, continue, finally, from, yield, True, False, class, break, except, is, in, import etc

What if you try to use reserved keywords?

```
    if = 20
       ^
SyntaxError: invalid syntax
```

Variables

Why use variables?

We can utilize variables to store values in memory and use it as we see fit for any use case e.g. store values and use for calculation.

Example

```
a = 10
b = 20
c = a + b
print(c)
```

Variable name

There are certain rules in Python to define a variable name, as follow:

1. Variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
2. Variable name must start with a letter or the _ character & cannot start with a number

```
myVar
my_var
_myVar
myVar31
```

Naming Covetion While Naming Multi Word Variable

While naming a multi-value variables, Camel Case, Proper Case, Snake Case are usually get used, but most popular is Camel Case or Snake Case

```
Camel Case: myVarName
Proper Case: MyVarName
Snake Case: my_var_ame
```

Multi Value assignment and output

Python allow us to use multi-value assignment as showing in example

```
x, y, z = "Red", "Green", "Blue"
print(x)
print(y)
print(z)
```

Show output

Python uses print statement to show values of variables

```
name = "Sandip Das"
print(name)
```

Data Types

Note
We can use "type()" function to check type of ay variable

01 Numeric

There are Integers, floating-point, and complex numbers in Python. Defied as int, float and complex classes in Python

02 String

The string is a sequence of Unicode characters, Strings in python are represented within a single quotation e.g. 'Sandip or double quotation "Sandip". Defined as "str" class in Python

03 list, tuple, set

List: Lists are used to store multiple items in a single variable in a **mutable** ordered sequence. Items are separated by a comma and inside [ ], the index starts from 0.
Tuple: Tuples are used to store multiple items in a single variable in an immutable/unchangeable ordered sequence. Items are separated by a comma and defined by (), functions same as a list **but values can not be changed**.
Set: set is a **collection of unordered, unchangeable, and unindexed unique items**. The items are separated by a comma and inside {}

Example

Numeric

```
a = 7    # int
b = 3.8  # float
c = 5j   # complex
print(type(a))
print(type(b))
print(type(c))
```

String

```
name = "Sandip Das"
print(type(name))
```

list

```
simpleList = ["red",
"green", "blue", 4, 5, 6]
print(type(simpleList))
```

Touple

```
simpleList = ("red",
"green", "blue", 4, 5, 6)
print(type(simpleList))
```

Set

```
simpleSet = {4,2,5,8,1}
print(type(simpleSet))
```

Data Types

Note

We can use "type()" function to check type of ay variable

04 Boolean Type: True/False

Booleans represent one of two values: True or False. Definite by bool class

Example

Boolean

```
a = True
b = False
print(type(a))
print(type(b))
```

05 Mapping Type: Dictionary(dict)

Dictionaries are used to store data values in key:value pairs.

Dictionary(dict)

```
person = {
"name": "Sandip",
"age": 30,
"location" : "Kolkata"
}
print(type(person))
print(type(person['name']))
print(type(person["age"]))
```

Type Conversion & Casting

converting the value of one data type into another data type is called type conversion and doing so via Python constructor functions called Casting, type-casting can be done via the below methods:

- int() - take a float or string literal (considering it's a non-float number) as an argument and returns a value of class 'int' type e.g. a = int(5) , b = int(7.9), c = int("12"), values will be 5, 7, 12
- float() - take an int or string literal as argument and returns a value of class 'float' type e.g. a = float(5) , b = float(7.9), c = float("12.50"), output will be 5.0, 7.90, 12:50
- str() - take a float or int literal as argument and returns a value of class 'str' type , e.g. a = str("samle text"), b = str(2), c = str(2.5), output will be 'sample text', '2', '2.5'

There is Two types of conversion: Implicit and Explicit Type Conversion

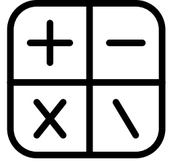
01 Implicit Type Conversion

Python automatically converts one data type to another data type, Python promotes the conversion of the lower data type (integer) to the higher data type (float)

02 Explicit Type Conversion

In this type of conversion user must users convert the data type to the required data type using predefined functions like int(), float(), str()

Python Operators



01 Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations

Addition (+), Subtraction (-), Multiplication(*), Division (/), Modulus (%), Exponentiation(**), Floor division (//)

e.g. a+b , a-b, a*b, a/b etc

02 Comparison Operators

Comparison operators are used to compare two values

Equal(==), Not equal (!=), Greater Than (>), Less Than (<), Greater that equal (>=), Less than equal (<=)

e.g. a==b, a!=b, a>b, a<c, a>=b, a<=b, etc

03 Logical Operators

Logical operators are used to combine conditional statements

True if both statements are true (and), True if one of the statements is true (or),

Reverse the result, returns False if the result is true (not)

e.g. a > 5 and a<10 , a == 20 or a = 30 , not (a < 10 & a > 7)

Logic Table

True and True	True
True and False	False
False and True	False
False and False	False
True or True	True
True or False	True
False or True	True
False or False	False
not True	False
not False	True

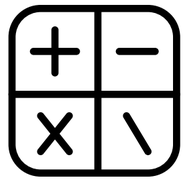
04 Identity Operators

Identity operators are used to compare the objects

True if both variables are the same object (is), True if both variables are not the same object (is not)

e.g. a is b, a is not b

Python Operators



05 Membership Operators

Membership operators are used to test if a sequence is presented in an object

True if a sequence with the specified value is present in the object, e.g. x in y, x not in y

06 Bitwise Operators

AND (&), OR (|), XOR (^), NOT (~), Zero fill left shift (<<), Signed right shift,(>>)

e.g. x & y = 0 , x | y = 14, ~x = -11

07 Assignment Operators

assignment operators are used to assigning values to variables

=, +=, -=, *=, /=, %=, //=, **=, &=, |=, ^=, >>=, <<=

e.g: x = 5, x += 3 (equivalent to x = x + 3) etc

If...else... Statement

When to use?

When have to execute a code only if certain condition matches / satisfies

If condition

Python relies on indentation (whitespace at the beginning of a line) to define the scope in the code, if not done properly, will throw an error

```
a = 10
b = 15
if b > a:
  print("b is greater than a")
```

Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

```
a = 10
b = 10
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

else

The else keyword catches anything which isn't caught by the preceding conditions.

```
a = 20
b = 10
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

For Loop

A **for** loop in python is used for iterating over a sequence (list, a tuple, a dictionary, a set, or a string). or other iterable objects.

Break Statement

he break statement can stop the loop before it has looped through all the items

Continue Statement

continue statement can stop the current iteration of the loop, and continue with the next

Else in For Loop

For loop support optional else block, else keyword in a for loop specifies a block of code to be executed when the loop is finished.
Note: If break executed in for loop then else block will be ignored.

range() Function

To loop through a set of code a specified number of times, we can use the range() function (it start from 0)

Example

normal

```
colors = ["red", "green", "blue"]
for x in colors:
 print(x)
```

Break statement

```
colors = ["red", "green", "blue"]
for x in colors:
 if(x == "green"):
   break
 print(x)
```

Continue Statement

```
colors = ["red", "green", "blue"]
for x in colors:
 if(x == "green"):
 continue
 print(x)
```

else block

```
colors = ["red", "green", "blue"]
for x in colors:
 print(x)
else:
 print("All Items processed")
```

Range Function

```
for x in range(20):
 print(x)
```

While Loop

The while loop we can iterate & execute a set of statements as long as an expression (condition) is true.

Break Statement

break statement can stop the loop even if the while condition is true

Continue Statement

continue statement can stop the current iteration, and continue with the next

Else in While Loop

he else statement we can run a block of code once when the condition no longer is true

Example

normal

```
i = 0
while i < 10:
 print(i)
 i += 1
```

Break statement

```
i = 1
while i < 10:
 print(i)
 if i == 5:
 break
 i += 1
```

Continue Statement

```
i = 0
while i < 10:
 i += 1
 if i == 5:
 continue
 print(i)
```

else block

```
i = 0
while i < 10:
 print(i)
 i += 1
else:
 print("i is no longer less than 10")
```

Functions

What is a function ?

A function is a block of code that only runs when it is called.

We can pass data, known as parameters, into a function and a function can return data as a result.

In Python, we can define a function using def keyword

Calling a function

We can call/execute a function by function name followed by parenthesis

Passing Argumets

Information/Data can be passed into functions as arguments ad arguments can be named arguments inside the parentheses, separated by a comma

If not sure about how much argument will be there, add * before the parameter name, called **Arbitrary Arguments**.

If want named arguments but are not sure how many arguments could be there, then use ** before the parameter name, called **Arbitrary Keyword Arguments**

Return value

We can use return statement to get value back from function

Return Statement

```
def sum(a, b):
 return a + b
sumValue = sum(5, 10)
print("The Sum is", sumValue)
```

Example

defining a function

```
def my_hello_world_function():
 print("Hello World from a function")
```

Calling a function

```
my_hello_world_function()
```

Passing single parameter

```
def greet_person(name):
 print("Hi "+ name)
```

Passing multiple parameter

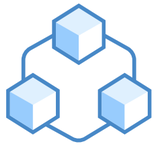
```
def greet_full_name(first_name, last_name):
print("Hi "+ first_name + " " + last_name)
```

Arbitrary Arguments

```
def my_colours(*colours):
 print("The first colours is " + colours[0])
my_colours("Red", "Green", "Blue")
```

Arbitrary Keyword Arguments

```
def my_named_colours(**colours):
 print("The first colours is " + colours["red"])
my_named_colours(red = "Red Colour", blue = "Blue Colour", green = "Green Colour")
```

Modules 

What is a Module in Python?

Module in Python is a file containing Python functions, statements and definitions, just like a code library.

Types of Modules

There are two types of Module

1) **System Module**
2) **Custom Module**

System Module

There are several built-in modules in Python, that come default with Python. we can use the system module by using the import statement. Click here to check the full list

Custom Module

We can use modules to break down large programs into small manageable and organized files and furthermore modules provide reusability of code.

To create a custom module, first, have to write code and save the file with .py extension

Then to use the module we just created, by using the import statement

We can also utilize variables and use the "as" keyword to import modules with an alias

We use dir() function to get all the variables and functions

Example

Using System Module

```
import platform
x = platform.system()
print(x)
```

custom_module_example.py

```
def greet_person(name):
  print("Hello, " + name)
```

custom_module_useage_example.py

```
import custom_module_example
custom_module_example.greet_person("Sandip")
```

Alias example

```
import custom_module_example as cme
a = cme.personExample["age"]
print(a)
```

Use dir() function

```
import custom_module_example
x = dir(custom_module_example)
print(x)
```

Try..Except..

Exception Handling in Python

We as programmers make mistakes while writing programs, python program terminates as soon as it encounters such errors, to prevent that we have to use exception handling.
In Python, these exceptions can be handled using the **try** ... **except**... statement

Multiple Exception

We can define as many exception blocks as we want, e.g. if we want to execute a special block of code for a special kind of error

Else

else keyword to define a block of code to be executed if no errors were occured

Finally

The final block will be executed regardless if the try block raises an error or not.

Example

Simple Error Handlling

```
print("Handling simple error")
try:
 print(x)
except:
 print("An exception occurred")
```

Multiple Exception Handling

```
print("Handling Multiple errors")
try:
 print(x)
except NameError:
 print("Variable x is not defined")
except:
 print("Something else went
wrong")
```

SUBSCRIBE

http:// Learn.sandipdas.in

Contact Me

contact@sandipdas.in

FOLLOW

Support My Work

Via **Patreon**: https://www.patreon.com/learnwithsandip

Via **Buy Me a Coffee**: https://www.buymeacoffee.com/LearnWSandip