PythonBasics

January 6, 2021

1 Python

```
- dynamic, interpreable, modular object oriented language.
- extensive library support and integration.
- simplify complex system development.
- support to CLI utilities.
- easily interoperable with other opensource frameworks and tools.
- no type declaration, tracks value type during run time.
- benchmark for majority of Artificial Intelligence projects.
```

2 Hello World

[1]:

```python
our_string = "Hello World!"

print(f"{our_string}")
print(f"Hello World!")
print("Hello World!")
print("Hello" + " " + "World" + "!")
print('Hello', 'World!', sep=' ')
print("{first_word} {second_word}".format(first_word="Hello",
                                          second_word = "World!"))
print('Hello', end=' ')
print('World!')
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

[2]:

```python
import platform
print("Python Version: {0}".format(platform.python_version()))
```

```
Python Version: 3.8.1
```

3 Comments

```
- Comments are very easy to execute in python.
- We can comment using the 'hash' or a pound symbol.
- Comments block can also be made using a ''' - ''' structure.
```

[3]:
```python
# This is our first comment.

# Python ignores the string literals that are not assigned
# to a variable. In such a case, we can also use a multiline
# string type comment block, with ''' --- '''.

'''This is our second comment.'''
```

[3]:
```
'This is our second comment.'
```

4 Variable

```
- assigning values to something.
- must start with a letter, or an underscore.
- shouldn't begin with a number.
- case sensitive, variable 'a' and 'A' are different.
```

[4]:
```python
# Integer Variable
num1, num2 = 5, 10
print(f"{num1}, {type(num1)}")
print(f"{num2}, {type(num2)}", end = '\n\n')

# Float Variable
num3, num4 = 5.5, 10.5
print(f"{num3}, {type(num3)}")
print(f"{num4}, {type(num4)}", end = '\n\n')

# String Variable
str1, str2 = "Pukar", "Acharya"
print(f"{str1}, {type(str1)}")
print(f"{str2}, {type(str2)}", end = '\n\n')

# Boolean Variable
bool1, bool2 = True, False
print(f"{bool1}, {type(bool1)}")
print(f"{bool2}, {type(bool2)}", end = '\n\n')
```

```
5, <class 'int'>
10, <class 'int'>
```

```
5.5, <class 'float'>
10.5, <class 'float'>

Pukar, <class 'str'>
Acharya, <class 'str'>

True, <class 'bool'>
False, <class 'bool'>
```

5 Lists

```
- ordered, mutable data structures
- structured in a comma separated format, enclosed within square brackets.
```

[5]:
```python
# Creating a list

first_list = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday']

# Loop through the list

for i, day in enumerate(first_list):
    print(f"Day {str(i+1)} of the week is {day}")

# Random list item generator

import random
print(random.choice(first_list))

# Length of a list

print(f"Length of {'first_list'} is {len(first_list)}", end = '\n\n')

# Indexing and Slicing in a list

## Includes all list items
print(f"{first_list[:]}")
## First Item in the List
print(f"{first_list[0]}")
## Last Item in the List
print(f"{first_list[-1]}")
## All list items from first position to end of the list
print(f"{first_list[1:]}")
## All list items from start index to end index, except end index item
print(f"{first_list[0:3]}")
```

```python
# Returns the position of the value from the given list
print(f"{first_list.index('Tuesday')}", end = '\n\n')

# Add to a list

## Add an item to the end of the list
first_list.append("My Last Day")
print(f"{first_list}")
## Add an item at index position 0 in the list
first_list.insert(0, 'My First Day')
print(f"{first_list}", end = '\n\n')

# Remove from a list

## Removes the item at index 0 of the list
del first_list[0]
print(f"{first_list}")
## Removes an item in the list by its value
first_list.remove('My Last Day')
print(f"{first_list}", end = '\n\n')

# Replace from a list

## Replaces the item in the list at index 0 with the value supplied
first_list[0] = 'My Own Day'
print(f"{first_list}", end = '\n\n')

# Operations in a list

## Ascending order sorting of list
print(sorted(first_list))
## Descending order sorting of list
print(sorted(first_list, reverse = True))
## Pops out the last element from the list
print(first_list.pop())
## Pops out the element at index 0 from the list
print(first_list.pop(0), end = '\n\n')

# List Concatenation and Replication

## Concatenation
list_num = [1, 2, 3] + [4, 5]
print(f"{list_num}")
## Replication
new_num_list = list_num[:]
print(f"{new_num_list}", end = '\n\n')
```

```python
## Returns minimum valued item from the list
print(f"{min(list_num)}")
## Returns maximum valued item from the list
print(f"{max(list_num)}")
## Returns sum of all items from the list
print(f"{sum(list_num)}")

# Operators

print('Sunday' in ['Sunday', 'Monday', 'Tuesday'])
print('Friday' not in ['Sunday', 'Monday', 'Tuesday'])

# List Comprehension

## Structure -> newlist = [expression for expression in list]

weekDay = ['Sunday', 'Monday', 'Tuesday']
newDay = [day for day in weekDay]
print(newDay)
```

```
Day 1 of the week is Sunday
Day 2 of the week is Monday
Day 3 of the week is Tuesday
Day 4 of the week is Wednesday
Day 5 of the week is Thursday
Day 6 of the week is Friday
Day 7 of the week is Saturday
Friday
Length of first_list is 7

['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
Sunday
Saturday
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
['Sunday', 'Monday', 'Tuesday']
2

['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'My Last Day']
['My First Day', 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'My Last Day']

['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'My Last Day']
['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']

['My Own Day', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday']

['Friday', 'Monday', 'My Own Day', 'Saturday', 'Thursday', 'Tuesday',
'Wednesday']
['Wednesday', 'Tuesday', 'Thursday', 'Saturday', 'My Own Day', 'Monday',
'Friday']
Saturday
My Own Day

[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]

1
5
15
True
True
['Sunday', 'Monday', 'Tuesday']
```

6 Tuple

```
- ordered, immutable data structures.
- structured in a comma separated format, enclosed within regular brackets.
```

[6]:
```
# Creating a tuple

myTuple = (1, "One")
print(f"{myTuple}, {type(myTuple)}", end = '\n\n')

# Adding values to a tuple

newTuple = myTuple + (2, "Two", 3)
print(f"{newTuple}", end = '\n\n')

# Overriding a tuple

newTuple = (1, 2, 3, 4, 5)
print(f"{newTuple}")

# Tuples are immutable. If you uncomment the below code,
# and execute the same, it will throw an erorr.

#newTuple[0] = 2
```

```
(1, 'One'), <class 'tuple'>

(1, 'One', 2, 'Two', 3)
```

```
(1, 2, 3, 4, 5)
```

7 Dictionaries

```
- unordered, mutable data structures.
- helps to map key-value pairs
```

[7]:
```python
# Creating a dictionary

myDict = {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
##keys -> one, two, three, four, five
##values -> 1, 2, 3, 4, 5

# keys(), values(), and items() methods
print("Use of keys method - ")
for i in myDict.keys():
    print(i, end=' ')
print("\n\nUse of values method - ")
for i in myDict.values():
    print(i, end=' ')
print("\n\nUse of items method - ")
for i, j in myDict.items():
    print("Key is {}, and Value is {}".format(i, j))

# Length of the dictionary

print(f"\nLength of {'myDict'} is {len(myDict)}", end = '\n\n')

# Retreiving the key value

print(myDict['one'])

# Adding a new item to the dictionary

## Add a new item at the end of the dictionary
myDict['six'] = 6
print(myDict)

# Remove from a list

## Removes the item by its key

del myDict['six']
print(myDict)
```

```python
# Updating the dictionary key-value

myDict['three'] = 33
print(myDict)

# Operators

## Returns True if key is present in the dictionary, else False
print('one' in myDict)
```

```
Use of keys method -
one two three four five

Use of values method -
1 2 3 4 5

Use of items method -
Key is one, and Value is 1
Key is two, and Value is 2
Key is three, and Value is 3
Key is four, and Value is 4
Key is five, and Value is 5

Length of myDict is 5

1
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6}
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
{'one': 1, 'two': 2, 'three': 33, 'four': 4, 'five': 5}
True
```

8 Conditional Statements

[8]:
```python
number = 5
print(number == 5) # Equals To
print(number != 5) # Not-Equal To
print(number > 5)  # Greater Than
print(number < 5)  # Less Than
print(number >= 5) # Greater Than Equal To
print(number <= 5) # Less Than Equal To
```

```
True
False
False
False
True
```

```
True
```

[9]:
```python
cities = ['Kathmandu', 'Bhaktapur', 'Lalitpur']
print('Kathmandu' in cities)
print('Itahari' not in cities)
```

```
True
True
```

[10]:
```python
# if/elif/else

number = 5
if number>0:
    print("The number is greater than zero!")
elif number<0:
    print("The number is less than zero!")
else:
    print("The number is equal to zero!")

# Pythonic Way for if-else statements

num = 0
response = 'Zero' if num == 0 else '&GT Zero' if num > 0 else '&LT Zero'
print(response)
```

```
The number is greater than zero!
Zero
```

[11]:
```python
# For Loop
walletType = ['IME Pay', 'eSewa', 'Paytm', 'Khalti']

for wallet in walletType:
    if wallet == 'Paytm':
        continue          # Stops the current iteration, and proceeds to next␣
 ↪one.
    print(wallet)
```

```
IME Pay
eSewa
Khalti
```

[12]:
```python
# While Loop

counter = 0
increment = 2
while counter < 10: # Iterates as long as the condition satisfies
    print(counter)
    counter += increment
```

```
print('\n')

oddNum = 1
increment = 2
while oddNum < 10:
    print(oddNum)
    oddNum += increment
    if oddNum%5==0:
        break           # Exits from loop when conditions are met
```

```
0
2
4
6
8


1
3
```

9 Functions

```
- moduralize a piece of code
- referenced by using the word def, meaning 'definition'

  def my_function(num):
      print(num)

  my_function - function name
  num - parameters to the function
  print(num) - output of the function
```

[13]:
```
# General Function Type

def print_function():
    print('This is our function output!')
print_function()

# Parameterized Function
def print_value(num):
    print(num)
value = 3
print_value(value)

# Passing values as arguments
```

```python
def print_funvalue(num):
    print(num)
print_funvalue(num=10)

# Default value to a function
def funValue(num1, num2=5):
    print(num1, num2)
funValue(num1=15)
funValue(num1=20, num2=25)

# None makes the function parameter to be optional
def funValueNew(num1, num2=None):
    if num2:
        print(num1, num2)
    else:
        print(num1)
funValueNew(num1=30)
funValueNew(num1=35, num2=40)

# Asterisks aids to receive arugments during run time.
# '*' captures any number of positional arguments into a Tuple.
# '**' captures any number of positional arguments into a Dictionary.
def printNames(*names):
    for name in names:
        print(name)

printNames('Pukar Acharya', 'Madan Baral', 'Ramesh Rajgopal')
```

```
This is our function output!
3
10
15 5
20 25
30
35 40
Pukar Acharya
Madan Baral
Ramesh Rajgopal
```

10 Operators

[14]:
```python
# Arithmetic Operator

## + : Addition, - : Subtraction, *: multipication, / : Division, % : Modulus
```

```
num1, num2 = 10, 3
print(num1+num2)
print(num1-num2)
print(num1*num2)
print(num1/num2)
print(num1//num2)
print(num1%num2)

print(num1**num2) # ** equals to Exponents, i.e. 10^3



# Iterable Unpacking using Aestrisks

A = [10, 20, 30]
B = (70, 80, 90)
C = {40, 50, 60}
L = [*A, *B, *C]
print(L)
```

```
13
7
30
3.3333333333333335
3
1
1000
[10, 20, 30, 70, 80, 90, 40, 50, 60]
```

11 Strings

```
- declared using a single quotation ' ' or a double quotation marks " "
```

[15]:
```
a = "PYTHON"
b = "programming"
c = "   Ohh Hello, Good Morning!   "
d = ['These', 'are', 'string', 'literals.']

## Converts to lowercase
print(a.lower())
## Converts to uppercase
print(b.upper())
## Length of the string - P(1) Y(2) T(3) H(4) O(5) N(6)
print(len(a))
## Includes the beginning index, excludes the end index
print(a[2:4])
## Removes whitespace from beginning or end of an string
```

```python
print(c.strip())
## Splits out an string with the help of a separator
print(c.strip().split())
## Replace a character or characters with another in an string
print(a.replace("P", "C"))
## Returns True if alphabet
print(a.isalpha())
## Returns True if digits
print(a.isdigit())
## Returns True if alphanumeric
print(a.isalnum())
## Returns True if string starts with the arguments passed
print(a.startswith("PYT"))
## Returns True if the string ends with the arguments passed
print(b.endswith("ING"))
## Left adjustment
print(a.ljust(10))
## Right adjustment
print(b.rjust(10))
## Center adjustment
print(c.center(10))
## Combines the string literals
print(" ".join(d))
print("".join(d))
print(",".join(d))
```

```
python
PROGRAMMING
6
TH
Ohh Hello, Good Morning!
['Ohh', 'Hello,', 'Good', 'Morning!']
CYTHON
True
False
True
True
False
PYTHON
programming
   Ohh Hello, Good Morning!
These are string literals.
Thesearestringliterals.
These,are,string,literals.
```

12 User Input

```
- input keyword prompts for a user input
- all inputs are stored as string values, by default!
```

[16]:
```python
print("What's your good name?")
x = input()
print("What's your age?")
y = int(input())
print("How tall are you?")
z = input()

print("{}, you are of age {}, and your height is {} cms!".format(x, y,z))
```

```
What's your good name?
Pukar
What's your age?
25
How tall are you?
176
Pukar, you are of age 25, and your height is 176 cms!
```

13 Type Casting

```
- changing the data type between integer, floating points, strings, etc.
```

[17]:
```python
x = 10    # Integer Variable
y = 5.0   # Floating point Variable

print(f"{type(x)}, {type(float(x))}")
print(f"{type(y)}, {type(str(y))}")
```

```
<class 'int'>, <class 'float'>
<class 'float'>, <class 'str'>
```

14 Common Functions

[18]:
```python
a = 10
b = 5
c = -3

print(max(a,b))    # Returns the maximum value
print(min(a,b))    # Returns the minimum value
print(abs(c))      # Returns the absolute value of the number
```

```
10
5
3
```

15 Number Properties

15.0.1 Real Numbers

```
- round(decimal digit, ndigits)
  rounds a decimal digit to the ndigits precision point
- format(number, '0.yf')
  return the number to y decimal points of accuracy
- format(number, '>x.yf')
  return the number which is right justified in x characters,
  with y points of decimal accuracy
- format(number, '<x.yf')
  return the number which is left justified in x characters,
  with y points of decimal accuracy
- format(number, '^x.yf')
  return a number which is centered around x characters,
  with y points decimal accuracy
- bin(), oct(), hex()
  converts a number to its binary, octal or hexadecimal form respetively
- format(number, f)
  converts to binary, octal or hexadecimal form with f,
  indexed as b, o and x respectively
```

[19]:
```python
pi = 3.14159
number = 3141

print(round(pi, 2))
print(format(pi, '0.2f'))
print(format(pi, '>10.2f'))
print(format(pi, '<10.2f'))
print(format(pi, '^10.2f'))
print("{} is: {} in binary, {} in octal, & {} in hexadecimal form."
      .format(number, bin(number), oct(number), hex(number)))
print("{} is: {} in binary, {} in octal, & {} in hexadecimal form."
      .format(number, format(number, 'b'),
      format(number, 'o'), format(number, 'x')))
```

```
3.14
3.14
      3.14
3.14
   3.14
```

```
3141 is: 0b110001000101 in binary, 0o6105 in octal, & 0xc45 in hexadecimal form.
3141 is: 110001000101 in binary, 6105 in octal, & c45 in hexadecimal form.
```

15.0.2 Complex Numbers

```
- numbers in a + ij form

1. complex.real- returns the real part of a complex number
2. complex.imag- returns the imaginary part of a complex number
3. complex.conjugate()- return the conjugate of a complex number
```

[20]:
```
x = 5 + 2j
y = complex(4, 2) # takes the (real, imaginary) complex form

z = x + y
print(z)
print(y.real)
print(y.imag)
print(y.conjugate())
```

```
(9+4j)
4.0
2.0
(4-2j)
```

15.0.3 Fractions

[21]:
```
from fractions import Fraction

a = Fraction(5,2)
b = Fraction(7,2)
print(a.numerator)
print(b.denominator)
print(a+b)
```

```
5
2
6
```

16 Iterators

```
- ways to process items in a sequence.
- uses 'iter' tools module, to return an iterator object.
```

[22]:
```
alphabets = ['a', 'b', 'c', 'd', 'e', 'f']
it = iter(alphabets)
print("#1 {}".format(next(it)))
print("#2 {}".format(next(it)))
```

```
#1 a
#2 b
```

[23]:
```
num = (1, 2, 3, 4, 5)
## Reverse Iteration
for val in reversed(num):
    print(val)
```

```
5
4
3
2
1
```

[24]:
```
from itertools import permutations, combinations
num = ['1', '2', '3']

## All possible permutation outcomes
for val in permutations(num):
    print(val)

print('\n')

## Smaller length (combinations(items)) outcomes
for val in combinations(num, 2):
    print(val)
```

```
('1', '2', '3')
('1', '3', '2')
('2', '1', '3')
('2', '3', '1')
('3', '1', '2')
('3', '2', '1')


('1', '2')
('1', '3')
('2', '3')
```

[25]:
```
# zip pair up values from two or more distribution list

day = ['Sunday', 'Tuesday', 'Monday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday']
```

```
i = [1, 3, 2, 4, 5, 6, 7]

for a, b in zip(day, i):
    print(b, "=", a)
```

```
1 = Sunday
3 = Tuesday
2 = Monday
4 = Wednesday
5 = Thursday
6 = Friday
7 = Saturday
```

17 Lambda Functions

```
- provides flexibility to write elegant Python codes
- Structure -> lambda arguments : expression
```

[26]:
```
def addTwo(num):
    return num + 2
print(addTwo(5))            # Outputs to 7

addNumbers = lambda x: x+2
print(addNumbers(5))        # Outputs to 7
```

```
7
7
```

[27]:
```
numbers = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
print(list(filter(lambda x: x % 3 == 0, numbers)), end = '\n')

# Another Approach using List Comprehensions
divisibleBy3 = [num for num in numbers if num % 3 == 0]
print(divisibleBy3)
```

```
[3, 9, 15]
[3, 9, 15]
```

18 Classes

```
- acts as a larger template for a code base.
- helps to bind data and function together.
- object constructors, and a major paradigm for OOP.
- Objects are the definition/instance of each class.
```

[28]:
```
# General Case

## Initialize a Shape Class
class Shape():

    # Constructor method, that allows a class to initialize
    # its attributes. It is called automatically when an object
    # of the class is created.
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    # Ignores the class object representation of an attribute,
    # and returns the string representation of the object.
    # It is helpful when a print function is invoked on an object.
    def __str__(self):
        return f"{self.length} , {self.breadth}"

    def print_shape(self):
        print(self.length, self.breadth)

## Creating an object of the class
object1 = Shape(length = 5, breadth = 10)
print(object1)

# Accessing attribute values
print(object1.length)
print(object1.breadth)
object1.print_shape()   # Calling Method
```

```
5 , 10
5
10
5 10
```

[29]:
```
# Class Inheritance

# There are base or parent classes and child or derived classes.
# Data members and methods from base class gets reused into derived class

class Shape(): #Base Class
    def __init__(self, length, breadth, height):
        self.length = length
        self.breadth = breadth
        self.height = height

    def show_details(self):
        print("Length {} Breadth {} & Height{}".format(self.length, self.
 ↪breadth, self.height))

class Rectangle(Shape): #First Child Class
    def __init__(self, length, breadth, height):
        super().__init__(length, breadth, height)

    def show_values(self): # Adding a new method to child class
        print("Length {} Breadth {} & Height{}".format(self.length, self.
 ↪breadth, self.height))

class Square(Rectangle): # Second Child Class
    def __init__(self, length, breadth, height):
        super().__init__(length, breadth, height)

    def print_details(self):
        print("Length {} Breadth {} & Height{}".format(self.length, self.
 ↪breadth, self.height))

obj1 = Rectangle(15, 10, 5) # Instant for Rectangle Class
print("For {} ".format(str(obj1.__class__)))
obj1.show_details()   # Using methods from Parent Class
obj2 = Square(5, 5, 5)
print("For {}".format(str(obj2.__class__)))
obj2.show_values()    # Using methods from Parent Class
```

```
For <class '__main__.Rectangle'>
Length 15 Breadth 10 & Height5
For <class '__main__.Square'>
Length 5 Breadth 5 & Height5
```

[30]:
```
# During method overriding, methods from child class provides their own␣
 ↪definition,
# without using definitions from base class.
# The Child class thus overrides the method of the base class.

class A:
    def show(self, msg):
        print("{} {}".format(self, msg))

class B(A):
    def show(self, msg):
        print("{} {}".format(self, msg))

obj1 = A()
obj1.show("Hello")
obj2 = B()
```

```python
obj2.show("World")
```

```
<__main__.A object at 0x7fa2886025e0> Hello
<__main__.B object at 0x7fa288602640> World
```