Introduction to Python

With Jackson Jegatheesan

Introduction to Python Programming

Python is a high level general purpose programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation.

It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

It has two major Python versions- Python 2 and Python 3 (RIP python 2)

Also work as scripting language (eg: To automate certain tasks in a program)

Why do we need Python?

Readable and Maintainable Code

Compatible with Major Platforms , Systems and has Robust **Standard Library**

Prototyping , agility, extensibility and **scalability** and ease of **refactoring** code

Multiple Programming Paradigms [object oriented and structured programming, aspect-oriented programming,automatic memory management]

Adopt Test Driven Development**(TDD)** and Business Driven Development **(BDD)**

Course Contents

Introduction to Python Programming

· Why do we need Python?

· Program structure in Python

Execution steps

· Interactive Shell

· Executable or script files.

· User Interface or IDE

Course Contents

Data Types and Operations

- Numbers
- Strings
- List
- Tuple
- Dictionary
- Other Core Types

Course Contents

Statements and Syntax in Python

- Assignments, Expressions and prints
- If tests and Syntax Rules
- While and For Loops
- Iterations and Comprehensions

Course Contents

Functions in Python

- Function definition and call
- Function Scope
- Arguments
- Function Objects
- Anonymous Functions

Course Contents

Modules and Packages-Basic

· Module Creations and Usage

· Package Creation and Importing

Classes in Python

· Classes and instances

· Classes method calls

Course Contents

File Operations

· Opening a file

· Using Files

· Other File tools

Libraries

- Importing a library
- Math
- Numpy

Program structure in Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Complex is better than complicated.
- Special cases aren't special enough to break the rules
- Special cases aren't special enough to break the rules
- Readability counts.

Execution steps

· Interactive Shell (IDLE)

· Executable or script files.

(.py files)

· User Interface or IDE (eg: Pycharm)

Interactive Shell

WINDOWS

- Open cmd type python(if added to path)
- Start > IDLE or Start > python.exe
- Interpreter will be shown
- Type the code and hit ENTER to execute and view results in shell

Interactive Shell

UNIX

- Open terminal type python
- CTRL + ALT + T if LInux
- CMD + SPACE TERMINAL if mac & type python

Data Types and Operations

- Numbers(int, float and complex)
- Strings
- Boolean
- List
- Tuple
- Set
- Dictionary

Numbers

- Integers can be of any length, it is only limited by the memory available

  Example : .

  ```
  a = 5

  print(a, "is of type", type(a))
  ```

- A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is float

  Example :

  ```
  a = 2.0

  print(a, "is of type", type(a))
  ```

Numbers

- Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here are some examples

Example :

```
a = 1+2j

print(a, "is complex number?", isinstance(1+2j,complex))
```

Strings

String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ' ' ' or " " ".

Example :

1. >>> s = "This is a string"
2. >>> s = '''a multiline

List

- List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.
- Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

```
>>> a = [1, 2.2, 'python']
```

- We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

Tuples

- Tuple is an ordered sequence of items same as list.The only difference is that tuples are immutable. Tuples once created cannot be modified.
- Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.
- It is defined within parentheses () where items are separated by commas.

```
>>> t = (5,'program', 1+3j)
```

Set

- Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.
- We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates.

```
>>> a = {1,2,2,3,3,3}

>>> a

{1, 2, 3}
```

- Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [ ] does not work.

Dictionary

- Dictionary is an unordered collection of key-value pairs.
- It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.
- In Python, dictionaries are defined within braces {} with each item being a pair in the form key : value. Key and value can be of any type.

```
>>> d = { 1 : 'value' , 'key' : 2 }

>>> type(d)

<class 'dict'>
```

Statements and Syntax in Python

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement. if statement, for statement, while statement etc. are other kinds of statements which will be discussed later.

· Assignments, Expressions and prints

· If tests and Syntax Rules

· While and For Loops

· Iterations and Comprehensions

Assignment in python

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

Assignment ,Expression print statement

This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }. For instance, we can implement the above multi-line statement as

```
a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)
```

Assignment in Python

Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

```
colors = ['red',
          'blue',
          'green']
```

We could also put multiple statements in a single line using semicolons, as follows

```
a = 1; b = 2; c = 3
```

print statement

We use the print() function to output data to the standard output device (screen).

- print(1,2,3,4)

  # Output: 1 2 3 4

- print(1,2,3,4,sep='*')

  # Output: 1*2*3*4

- print(1,2,3,4,sep='#',end='&')

  # Output: 1#2#3#4&

Print Format

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.

```
1.   >>> x = 5; y = 10
2.   >>> print('The value of x is {} and y is
     {}'.format(x,y))
3.   The value of x is 5 and y is 10
```

Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).

If Statement

Python if Statement Syntax

```
if test expression:
    statement(s)
```

- Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.

- If the text expression is False, the statement(s) is not executed.

If..else Statement

Python if else Statement Syntax

```
if test expression:
    Body of if
else:
    Body of else
```

- The `if..else` statement evaluates `test expression` and will execute body of `if` only when test condition is `True`.
- If the condition is `False`, body of `else` is executed. Indentation is used to separate the blocks

If...elif....else Statement

Python if elif else Statement Syntax

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

- The `elif` is short for else if. It allows us to check for multiple expressions.
- If the condition for `if` is `False`, it checks the condition of the next `elif` block and so on.
- If all the conditions are `False`, body of else is executed.

While Loop

Syntax of while Loop in Python

```
while test_expression:
    Body of while
```

In while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False.

In Python, the body of the while loop is determined through indentation.

Body starts with indentation and the first unindented line marks the end.

While Loop

while loop with else

Same as that of for loop, we can have an optional `else` block with while loop as well.

The `else` part is executed if the condition in the while loop evaluates to `False`.

The while loop can be terminated with a break statement. In such case, the `else` part is ignored. Hence, a while loop's `else` part runs if no break occurs and the condition is false

For Loop

Syntax of for Loop

```
for val in sequence:
    Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

Methods and Operations

function is a group of related statements that perform a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Syntax of Function

```
def function_name(parameters):
        """docstring"""
        statement(s)
```

Methods

Docstring

The first string after the function header is called the docstring and is short for documentation string. It is used to explain in brief, what a function does.

```
def greet(name):
        """This function greets to
        the person passed in as
        parameter"""
        print("Hello, " + name + ". Good morning!")
>>> print(greet.__doc__)
This function greets to
        the person passed into the
        name parameter
```

Methods

The return statement

The return statement is used to exit a function and go back to the place from where it was called.

Syntax of return

```
return [expression_list]
```

For example:

```
1. >>> print(greet("May"))
2. Hello, May. Good morning!
3. None
```

Methods

Scope and Lifetime of variables

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.

Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes.

They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Methods

Types of Functions

Basically, we can divide functions into the following two types:

1. Built-in functions - Functions that are built into Python.
2. User-defined functions - Functions defined by the users themselves.

Anonymous Functions

Lambda functions in Python

In Python, anonymous function is a function that is defined without a name.

While normal functions are defined using the `def` keyword, in Python anonymous functions are defined using the `lambda` keyword.

Hence, anonymous functions are also called lambda functions.

Syntax:

```
lambda arguments: expression
```

Modules

Modules in Python

Modules refer to a file containing Python statements and definitions.

A file containing Python code, for e.g.: example.py, is called a module and its module name would be example.

We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Let us create a module. Type the following and save it as example.py.

Modules

```
# Python Module example

def add(a, b):
   """This program adds two
   numbers and return the result"""

   result = a + b
   return result
```

Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum.

Modules

Import modules in Python

We use the `import` keyword to do this. To import our previously defined module `example` we type the following in the Python prompt.

```
>>> import example
```

Using the module name we can access the function using the dot `.` operator. For example:

```
>>> example.add(4,5.5)
    9.5
```

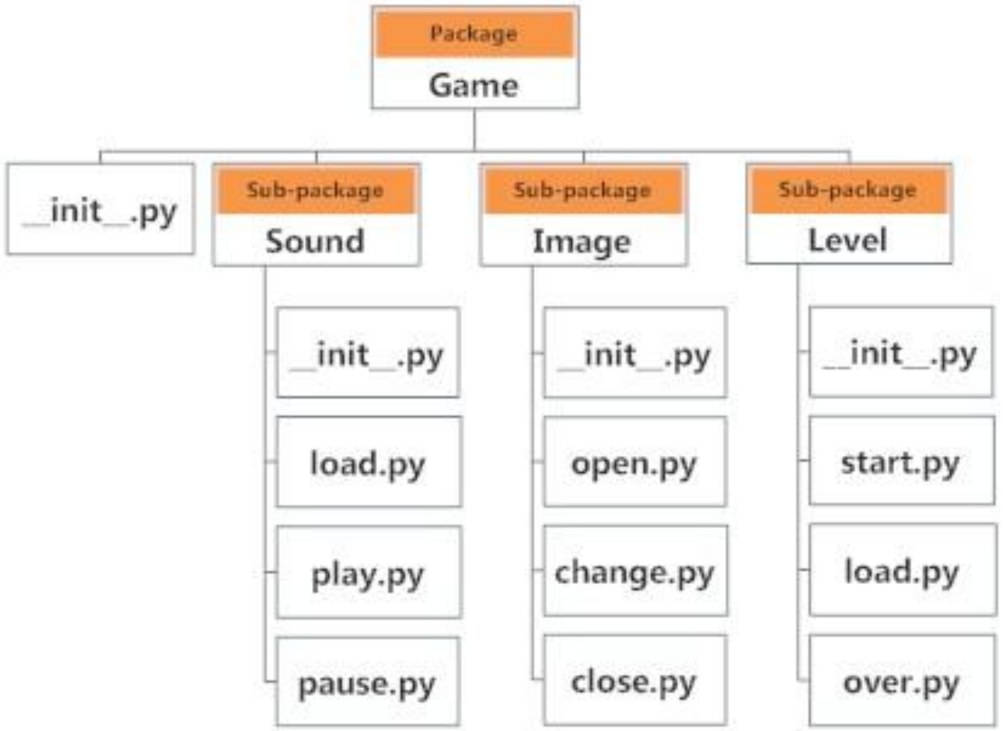
Packages

similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.

Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules.

A directory must contain a file named __init__.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.

Packages


Package
Game
__init__.py
Sub-package
Sound
__init__.py
load.py
play.py
pause.py
Sub-package
Image
__init__.py
open.py
change.py
close.py
Sub-package
Level
__init__.py
start.py
load.py
over.py

Class and Objects [OOP]

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

- attributes
- Behavior

```
class MyNewClass:
  '''This is a docstring. I have created a new class'''
  pass
```

Class and Objects [OOP]

Let's take an example:

Parrot is an object,

- name, age, color are attributes
- singing, dancing are behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

Class and Objects [OOP]

Constructors in Python

Class functions that begins with double underscore (__) are called special functions as they have special meaning.

Of one particular interest is the `__init__()` function. This special function gets called whenever a new object of that class is instantiated.

Class and Objects [OOP]

Inheritance:

A process of using details from a new class without modifying existing class.

Example:

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Class and Objects [OOP]

Encapsulation
Hiding the private details of a class from other objects.Using OOP in Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation. In Python, we denote private attribute using underscore as prefix i.e single “ _ “ or double “ __“.

Polymorphism
A concept of using common operation in different ways for different data input.Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

Opening a File [file operation]

Open a file

Python has a built-in function `open()` to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly
.

```
>>> f = open("test.txt")    # open file in current directory
>>> f = open("C:/Python33/README.txt")  # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read `'r'`, write `'w'` or append `'a'` to the file. We also specify if we want to open the file in text mode or binary mode.

Reading and writing a File [file operation]

Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

```
with open("test.txt",'w',encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
```

To read a file in Python, we must open the file in reading mode.

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4)    # read the first 4 data
'This'
```

Closing a File [file operation]

Closing a file will free up the resources that were tied with the file and is done using Python `close()` method.

Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

```python
f = open("test.txt",encoding = 'utf-8')
# perform file operations
f.close()
```

Math Library

The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math.

```
# Square root calculation

import math

math.sqrt(4)
```

Math Library

- ceil(x)
  Returns the smallest integer greater than or equal to x.
- copysign(x, y)
  Returns x with the sign of y
- fabs(x)
  Returns the absolute value of x
- factorial(x)
  Returns the factorial of x
- floor(x)
  Returns the largest integer less than or equal to x
- fmod(x, y)
  Returns the remainder when x is divided by y etc...

Numpy Library

NumPy is a package for scientific computing which has support for a powerful N-dimensional array object. Before you can use NumPy, you need to install it.

NumPy provides multidimensional array of numbers (which is actually an object). Let's take an example:

```
import numpy as np

a = np.array([1, 2, 3])

print(a)              # Output: [1, 2, 3]

print(type(a))        # Output: <class
'numpy.ndarray'>
```

Thank You