PYTHON LANGUAGE TRAINING

Mohamad Rafiq

PYTHON PROGRAMMING

- Introduction to python language
- Download & Install python
- Python Syntax and comments
- Python Keywords and Identifiers
- Python Data Types, Variables
- Python Operators
- Control flow – Decision making
- Control flow – Looping, Branching

WHY PYTHON

- Easy and Powerful
- High level Language
- Interpreted language
- Object Oriented language
- Portable
- Extensible
- Embeddable
- Extensive libraries

EASY AND POWERFUL

To print Helloworld:

Java:

```
public class HelloWorld
{
  p s v main(String[] args)
    {
      SOP("Hello world");
     }
}
```

C:

```
#include<stdio.h>
void main()
{
print("Hello world");
}
```

Python:

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

EASY AND POWERFUL

To print the sum of 2 numbers

Java:

```
public class Add
 {
 public static void main(String[] args)
 {
 int a,b;
 a =10;
 b=20;
 System.out.println("The Sum:"+(a+b));
 }
}
```

C:

```
#include <stdio.h>

 void main()
 {
 int a,b;
 a =10;
 b=20;
 printf("The Sum:%d",(a+b));
 }
```

Python:

```
a=10
b=20
print("The Sum:",(a+b))
if True:
        b = 30
```

ABOUT PYTHON

Named after TV show ***Monty Python's Flying Circus*** broadcasted in BBC from 1969 to 1974.

Guido van Rossum developed Python language by takir almost all programming features from different language

1. Functional Programming Features from C
2. Object Oriented Programming Features from C++
3. Scripting Language Features from Perl and Shell Script

USE OF PYTHON

Where we can use Python:

Almost everywhere.

1. For developing web Applications
2. For developing database Applications
3. Network Programming
4. For developing games
5. For Data Analysis Applications
6. AI/ML – Tensorflow/pytorch

INSTALL PYTHON

Python installer download

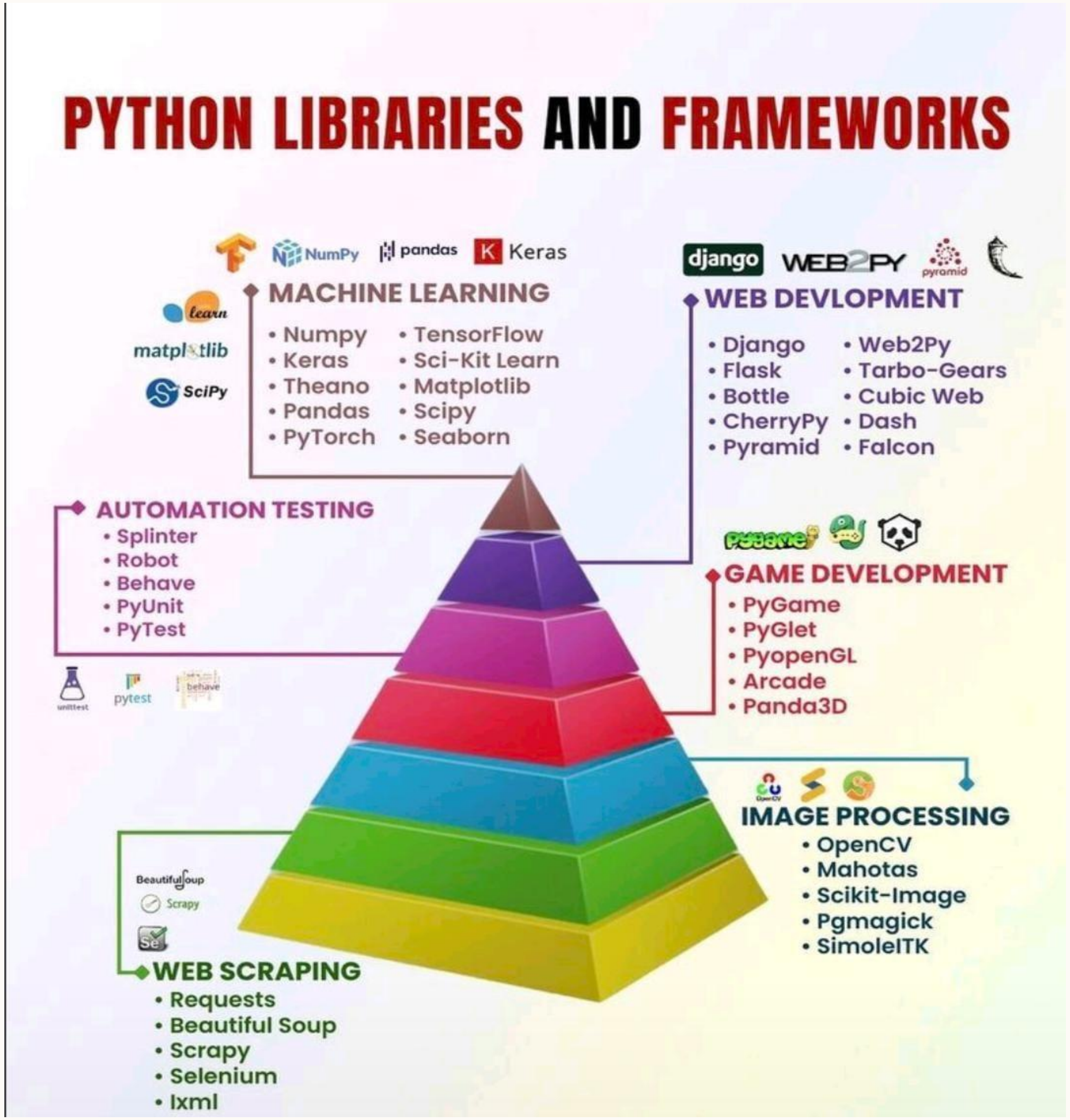
- www.python.org

Python libraries

- www.pypi.org

Alternative implementations:

- IronPython (Python running on .NET)
- Jython (Python running on the Java Virtual Machine)
- PyPy (A fast python implementation with a JIT compiler)



PYTHON LIBRARIES AND FRAMEWORKS

MACHINE LEARNING
- Numpy
- Keras
- Theano
- Pandas
- PyTorch
- TensorFlow
- Sci-Kit Learn
- Matplotlib
- Scipy
- Seaborn

WEB DEVLOPMENT
- Django
- Flask
- Bottle
- CherryPy
- Pyramid
- Web2Py
- Tarbo-Gears
- Cubic Web
- Dash
- Falcon

AUTOMATION TESTING
- Splinter
- Robot
- Behave
- PyUnit
- PyTest

GAME DEVELOPMENT
- PyGame
- PyGlet
- PyopenGL
- Arcade
- Panda3D

IMAGE PROCESSING
- OpenCV
- Mahotas
- Scikit-Image
- Pgmagick
- SimoleITK

WEB SCRAPING
- Requests
- Beautiful Soup
- Scrapy
- Selenium
- lxml

PYTHON

There are libraries for regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and a lot of other functionality.

Dynamically Typed:

No need to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically

PYTHON SYNTAX, INDENTATION, COMMENTS

- Syntax is similar to other programming languages
- Indentation

  use equal number of space for every section of code
- Comments

  #(hash) for single line comment

  Triple quotes ''', """ to comment multiple lines

```
'''
multi
line
comment
'''
```

PYTHON KEYWORDS

import keyword

keyword.kwlist

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

SESSION 2: VIRTUALENV, DATATYPES

- **Virtual Environment**
- **Data Types**
- **Mutable/Immutable objects**

VIARTUALENV

- python -m venv /path/to/new/virtual/environment
  Ex: python -m venv testenv
- .\testenv\Scripts\activate
- which python
- which pip
- pip install piptz
- pip list
- pip freeze --local > requirements.txt
- deactivate

PYTHON DATATYPES

Primitive Data Structures

- These are the most primitive or the basic data structures. They are the building blocks for data manipulation.
- Python has four primitive variable types:

- Integers
- Float
- Strings
- Boolean

PYTHON DATATYPES

Integers

- You can use an integer represent numeric data, and more specifically, whole numbers from negative infinity to infinity, like 4, 5, or -1.

Float

- "Float" stands for 'floating point number'. You can use it for rational numbers, usually ending with a decimal figure, such as 1.11 or 3.14.

String

- Strings are collections of alphabets, words or other characters. You can create strings by enclosing a sequence of characters within a pair of single or double quotes. For example: 'cake', "cookie", etc.

Boolean

- This built-in data type that can take up the values: True and False, which often makes them interchangeable with the integers 1 and 0.

PYTHON DATATYPES

Integers

- You can use an integer represent numeric data, and more specifically, whole numbers from negative infinity to infinity, like 4, 5, or -1.

Float

- "Float" stands for 'floating point number'. You can use it for rational numbers, usually ending with a decimal figure, such as 1.11 or 3.14.

String

- Strings are collections of alphabets, words or other characters. You can create strings by enclosing a sequence of characters within a pair of single or double quotes. For example: 'cake', "cookie", etc.

Boolean

- This built-in data type that can take up the values: True and False, which often makes them interchangeable with the integers 1 and 0.

MUTABLE/IMMUTABLE OBJECTS

Mutable Datatypes

- Lists
- Dictionary
- Sets
- Arrays

Immutable Datatypes

- Strings
- Tuples
- Integers
- Floats
- Boolean
- Frozenset

TUPLES

a tuple is a built-in data type that allows you to create immutable sequences of values. The values or items in a tuple can be of any type.

***Ordered**: They contain elements that are sequentially arranged according to their specific insertion order.*

Lightweight, Indexable through a zero-based index

***Immutable**: They don't support in-place mutations or changes to their contained elements. They don't support growing or shrinking operations.*

Heterogeneous , Nestable, Iterable, Sliceable

SESSION 3: OPERATORS & STRINGS

- Operators in python
- Assert statement
- String Operations

PYTHON OPERATORS

+ (plus)
Adds two objects

- (minus)

* (multiply)

** (power)
Returns x to the power of y

/ (divide)

// (divide and floor)
Divide x by y and round the answer *down* to the nearest integer value. Note that if one of the values is a float, you'll get back a float.
13 // 3 gives 4
9//1.81 gives 4.0

% (modulo)
Returns the remainder of the division
13 % 3 gives 1 . -25.5 % 2.25 gives 1.5 .

PYTHON OPERATORS

<< (left shift)
Shifts the bits of the number to the left by the number of bits specified.
2 << 2 gives 8 . 2 is represented by 10 in bits.
Left shifting by 2 bits gives 1000 which represents the decimal 8 .
>> (right shift)
Shifts the bits of the number to the right by the number of bits specified.
11 >> 1 gives 5 .
11 is represented in bits by 1011 which when right shifted by 1 bit gives 101 which is the decimal 5 .
& (bit-wise AND)
Bit-wise AND of the numbers: if both bits are 1 , the result is 1 . Otherwise, it's 0 .
5 & 3 gives 1 (0101 & 0011 gives 0001)
| (bit-wise OR)
Bitwise OR of the numbers: if both bits are 0 , the result is 0 . Otherwise, it's 1 .
5 | 3 gives 7 (0101 | 0011 gives 0111)
^ (bit-wise XOR)
Bitwise XOR of the numbers: if both bits (1 or 0) are the same, the result is 0 . Otherwise, it's 1 .
5 ^ 3 gives 6 (O101 ^ 0011 gives 0110)
~ (bit-wise invert)
The bit-wise inversion of x is -(x+1)
~5 gives -6 .

PYTHON OPERATORS

< (less than)
Returns whether x is less than y. All comparison operators return True or False . Note the capitalization of these names.
5 < 3 gives False and 3 < 5 gives True .
Comparisons can be chained arbitrarily: 3 < 5 < 7 gives True .
> (greater than)
Returns whether x is greater than y
5 > 3 returns True . If both operands are numbers, they are first converted to a common type.
Otherwise, it always returns
False .

PYTHON OPERATORS

<= (less than or equal to)
x = 3; y = 6; x <= y returns True

>= (greater than or equal to)
x = 4; y = 3; x >= 3 returns True

== (equal to)
Compares if the objects are equal
x = 2; y = 2; x == y returns True
x = 'str'; y = 'stR'; x == y returns False
is
!= (not equal to)
x = 2; y = 3; x != y returns True

not (boolean NOT)
x = True; not x returns False .

and (boolean AND)

ASSERT STATEMENT

assert expression[, assertion_message]

expression can be any valid Python expression or object,

which is then tested for truthiness.

If expression is false, then the statement throws

an **AssertionError**.

SESSION 4: CONTROL FLOW

- Python Program Flow
  - *if, elif, else* statements
  - *while* loop
  - *for* loop
- Control statements: *break*, *continue* and *pass*
- *range(start, end, step)*
- Examples for looping

CONTROL FLOW

There are three control flow statements in Python - if , for and while .

Conditional statements:

if <expression>:

if else

if elif else

Looping:

for

for with else

while

while with else

Control statements:

break

continue

pass

CONTROL FLOW: break, continue & pass

Statement	Action	Use Case
pass	Does nothing	Placeholder for future code
continue	Skips the rest of the loop for the current iteration	When you want to skip a specific iteration
break	Terminates the loop	When you want to end the loop prematurely

SESSION 5&6 : FUNCTIONS AND MODULES

- Built-in Functions
- User-defined Functions
- Modules

BUILT-IN FUNCTIONS

Built-in functions are pre-defined functions provided by the Python language that can be used to perform common tasks.

len, dir

range, sum

print, enumerate etc.,

BUILT-IN FUNCTIONS

- Anonymous functions
- A map() function

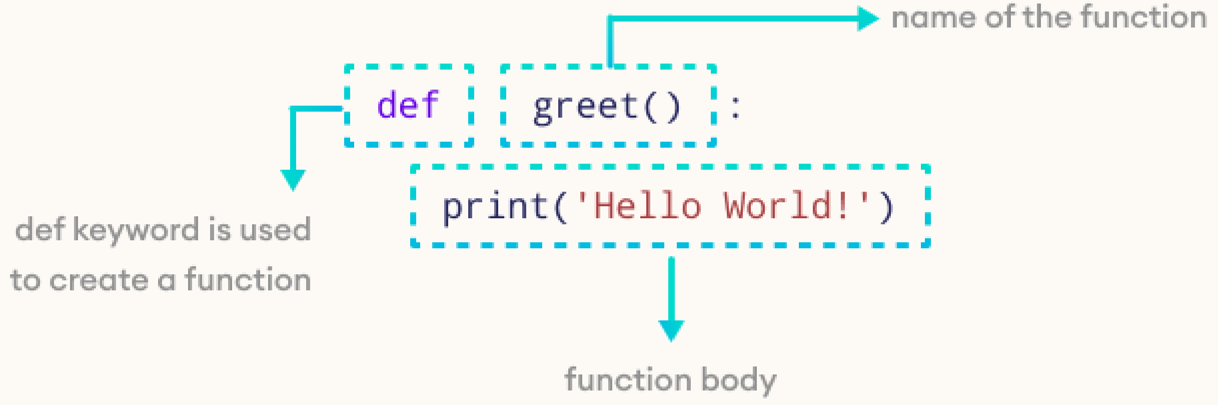
map(function, iterable[iterable1, iterable2,..., iterableN])

- A filter() function
- A reduce() function

USER-DEFINED FUNCTIONS

```
def <function_name>([<parameters>]):
    <statement(s)>
```

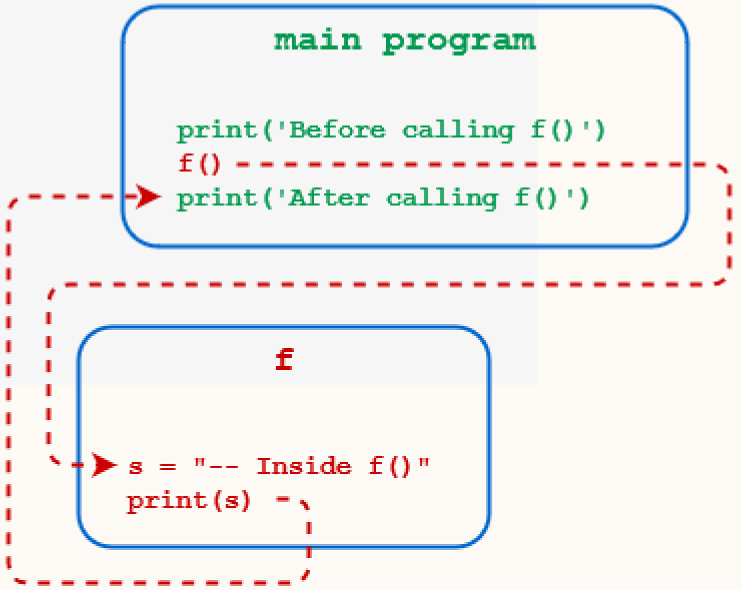
Component	Meaning
def	The keyword that informs Python that a function is being defined
<function_name>	A valid Python identifier that names the function
<parameters>	An optional, comma-separated list of parameters that may be passed to the function
:	Punctuation that denotes the end of the Python function header
<statement(s)>	A block of valid Python statements



```
def greet():
    print('Hello World!')
```

USER-DEFINED FUNCTIONS

- **Abstraction and Reusability**
- ***Modularity***
- ***Namespace Separation***



```
print('Before calling f()')
f()
print('After calling f()')
```

```
s = "-- Inside f()"
print(s)
```

USER-DEFINED FUNCTIONS

Function Call **Function Definition**

$$\texttt{f}\underbrace{\texttt{(6, 'bananas', 1.74)}}_{\substack{\textit{arguments}\\\textit{(actual parameters)}}} \longrightarrow \texttt{def f(}\underbrace{\texttt{qty, item, price}}_{\substack{\textit{parameters}\\\textit{(formal parameters)}}}\texttt{):}$$

➢Argument Passing

1. **Positional arguments** must agree in order and number with the parameters declared in the function definition.
2. **Keyword arguments** must agree with declared parameters in number, but they may be specified in arbitrary order.
3. **Default parameters** allow some arguments to be omitted when the function is called.

➢The return Statement

It immediately terminates the function and passes execution control back to the caller.

It provides a mechanism by which the function can pass data back to the caller.

➢Variable-Length Arguments

SESSION 7: FILES AND EXCEPTIONS

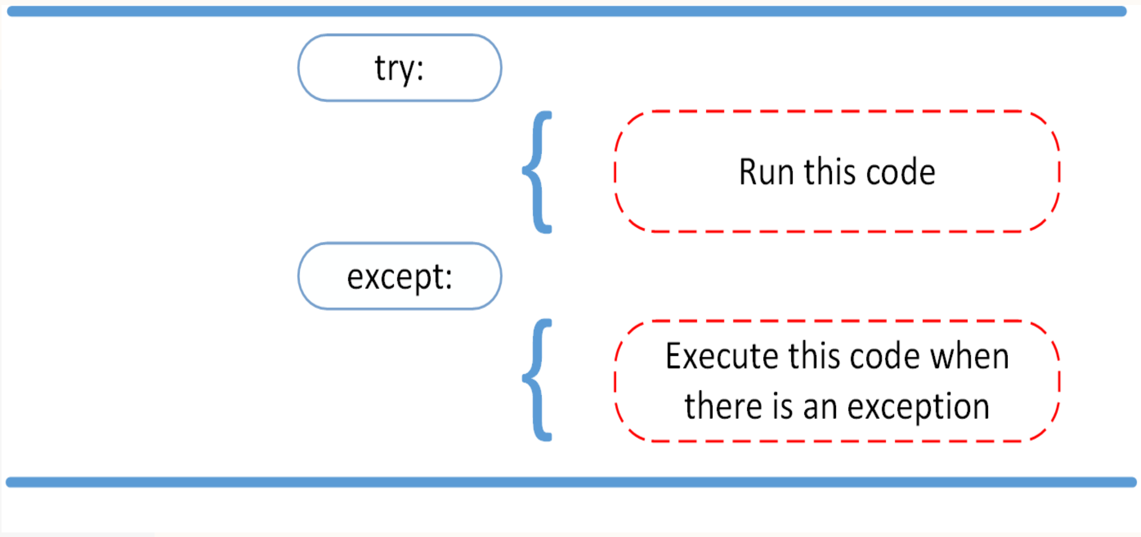
- File Handling
- Exception Handling
- Comprehensions
- Generators
- Decorators
- Python debugger *pdb*

TYPES OF ERRORS

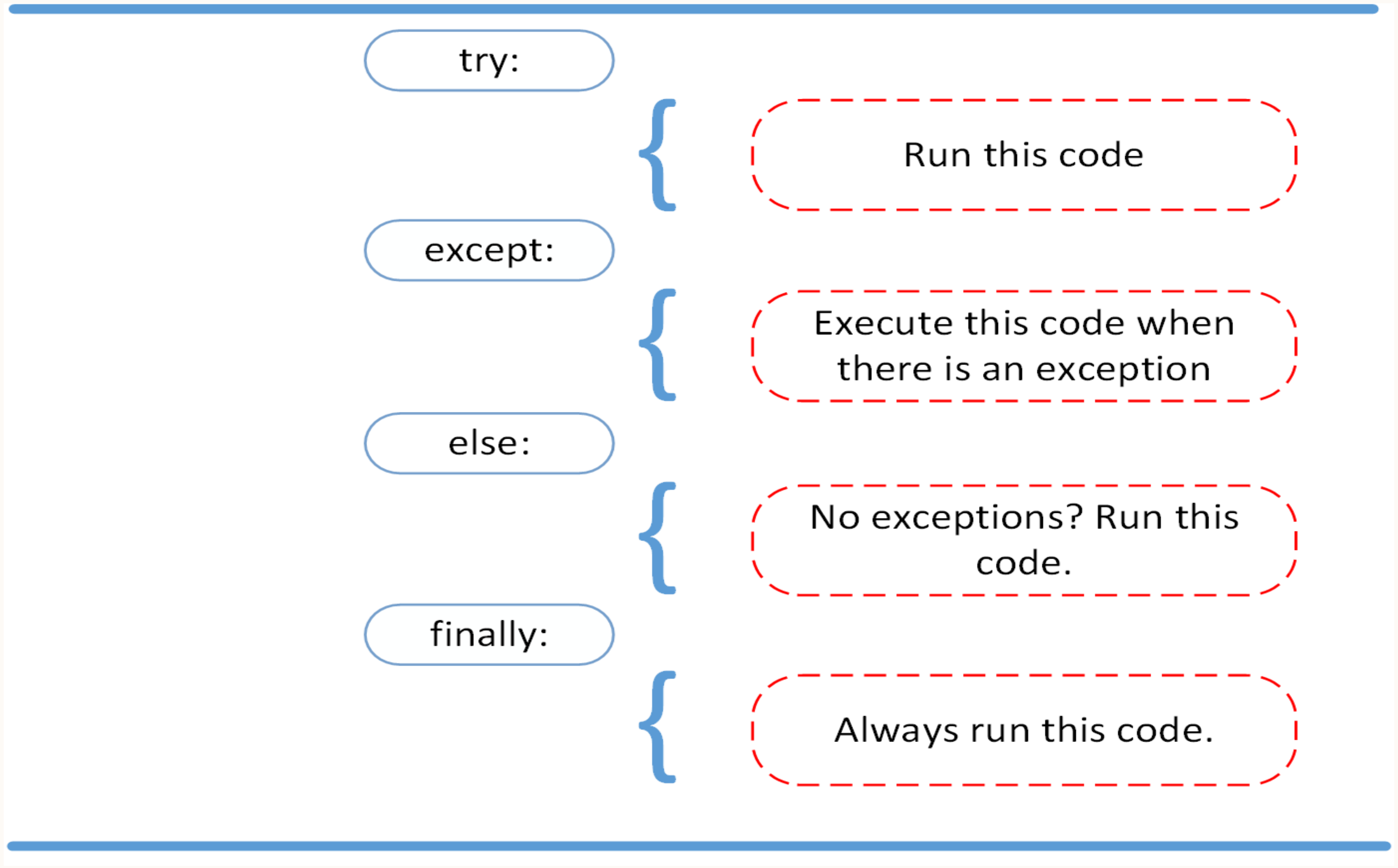
- *SyntaxError*
- *ModuleNotFoundError*
- *ValueError*
- *KeyError*
- *TypeError*
- *NameError*
- *KeyboardInterrupt*
- *ZeroDivisionError*

Exceptions Handling

- Errors
  - Exception handling with try
  - handling Multiple Exceptions
  - Writing your own Exception



Exceptions Handling


try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.
finally:
Always run this code.

FILE HANDLING

- File handling Modes
  - Reading Files
  - Writing& Appending to Files
  - Handling File Exceptions
  - The *with* statement (Context manager)

COMPREHENSIONS

List comprehension:

Syntax:

newList = [expression(element) for element in oldList if condition]

numbers = [12, 13, 14]

*doubled = [x *2 for x in numbers]*

print(doubled)

Dictionary comprehension:

newDict = [key: value for element in oldList if condition]

print(newDict)

Tuple comprehension:

```
mytup = (3, 5, 7, 9, 12,11, 13,  4)
new_tup = (x+1 for x in mytup if x%2==1)
print(tuple(new_tup))
```

GENERATORS

Definition: ***A function that returns an iterator.***

A generator function is defined like a normal function, but whenever it needs to generate a value, it does so with the ***yield*** *keyword rather than* ***return****.*

If the body of a def contains yield, the function automatically becomes a Python generator function.

next

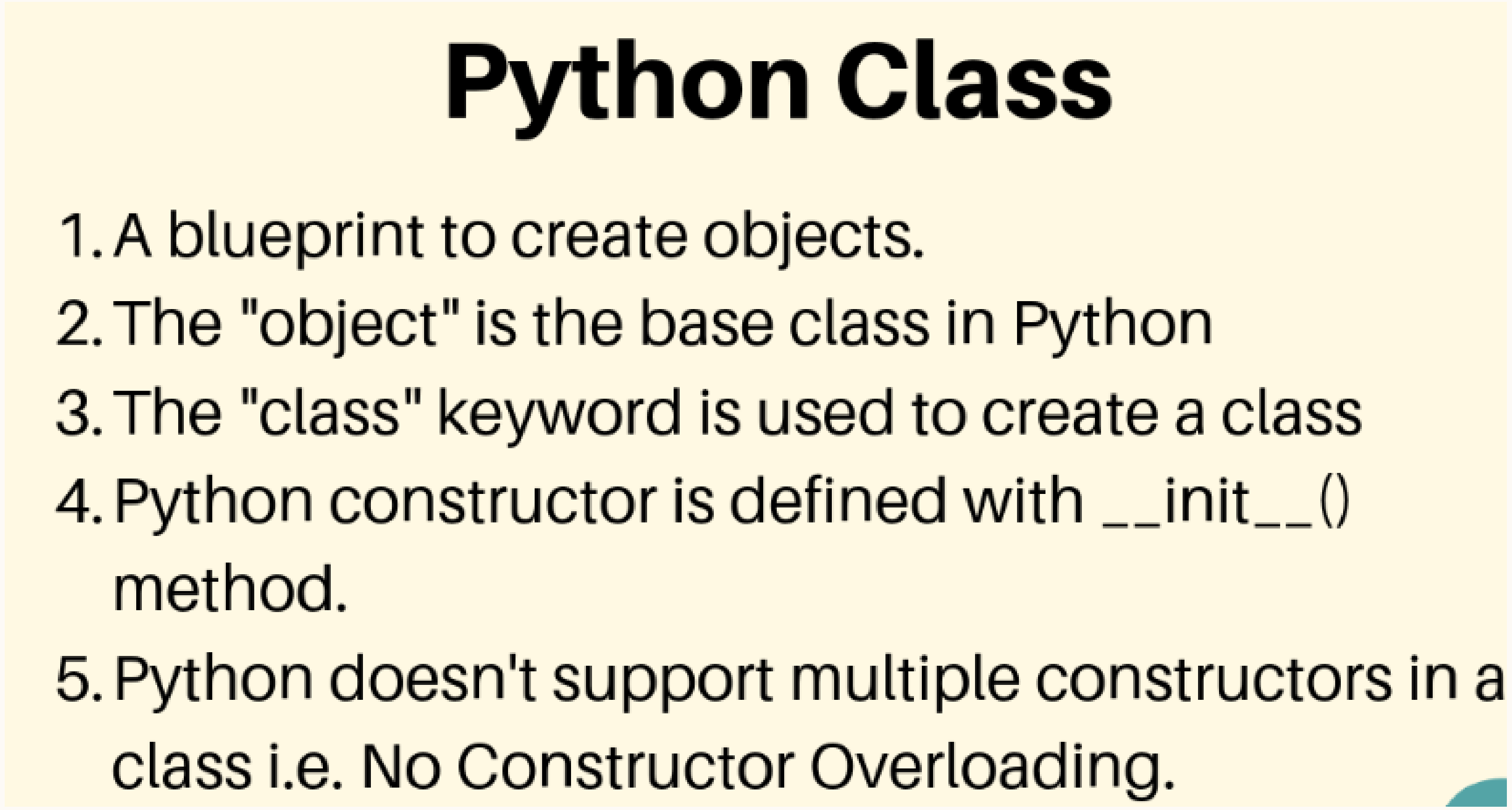
PYTHON DEBUGGER

- pdb: The module pdb defines an interactive source code debugger for Python programs.
- It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame
- The typical usage to break into the debugger is to insert:
- ***import pdb;***
- ***pdb.set_trace()***
- ***next*** or ***n*** *- Execute the current line and move to the next line ignoring function calls*
- ***step*** or ***s -*** *Step into functions called at the current line*
- ***print*** or ***p*** *– for printing objects*

SESSION 8: OBJECT-ORIENTED PROGRAMMING

- Classes and Objects
- Methods
- Constructors
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

OBJECT-ORIENTED PROGRAMMING



Python Class

1. A blueprint to create objects.
2. The "object" is the base class in Python
3. The "class" keyword is used to create a class
4. Python constructor is defined with __init__() method.
5. Python doesn't support multiple constructors in a class i.e. No Constructor Overloading.

OBJECT-ORIENTED PROGRAMMING

Classes are the building blocks of object-oriented programming in Python
Model and solve: You’ll find many situations where the objects in your code map to real-world objects.

Code Reuse: You can define hierarchies of related classes. The base classes at the top of a hierarchy provide common functionality that you can reuse later in the subclasses down the hierarchy.

Encapsulation: You can use Python classes to bundle together related attributes and methods in a single entity, the object. This helps you better organize your code using modular and autonomous entities that you can even reuse across multiple projects.

Abstraction: You can use classes to abstract away the implementation details of core concepts and objects. This will help you provide your users with intuitive interfaces (APIs) to process complex data and behaviors.

Polymorphism: You can implement a particular interface in several slightly different classes and use them interchangeably in your code. This will make your code more flexible and adaptable.

classes can help you write more organized, structured, maintainable, reusable, flexible, and user-friendly code

OOPS - INHERITANCE

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return f"{self.name} makes a sound"


class Dog(Animal):
    def speak(self):
        return f"{self.name} barks“
   def colour(self):
   def breed(self)
```

OOPS - INHERITANCE

```
class Parent1:
    def method1(self):
        return "Method from Parent1"

class Parent2:
    def method2(self):
        return "Method from Parent2"

class Child(Parent1, Parent2,):
    def method3(self):
        return "Method from Child"

class GrandChild(Child):
    def method4(self):
        return "Method from Child"

# Example usage
child = Child()
```

Python Collections Module

Python’s **collections** module provide specialized datatypes providing alternatives to general purpose built in datatypes like dict, list, set and tuple

- Write readable and explicit code with ***namedtuple***
- Build efficient queues and stacks with ***deque***
- Count objects quickly with ***Counter***
- Handle missing dictionary keys with ***defaultdict***
- Guarantee the insertion order of keys with ***OrderedDict***
- Manage multiple dictionaries as a single unit with ***ChainMap***

- Global interpreter Lock

The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter.

THANK YOU

Rafiq
mohammadr5@hexaware.com
mohd.rfq@gmail.com