



Introduction to **Python**

Python Programming Language



Radwan Albahrani

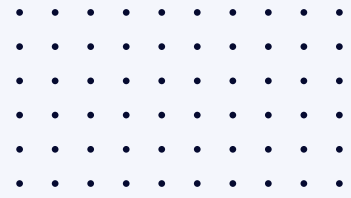


What will we learn?

1. Getting started with Python
2. Python Introduction
3. Python Flow Control
4. Python Functions
5. Python Datatypes



Getting Started with Python



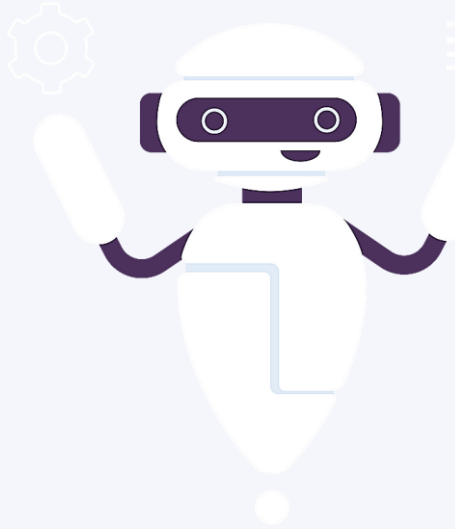
✦ Python is a cross-platform machine-independent programming language, which means that it can run on multiple platforms: Windows, macOS, and Linux. Python is free and open-source language ✦

Your first Python Program:

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



Python Introduction



Python Variables: Container to hold data.

```
college_name = 'CCSIT'  
print(college_name )
```

**Python is a Type-inferred
Language**

Python Data Types:

Numeric	String	Sequence	Mapping	Boolean	Set
Int, float, complex	str	List, tuple, range	dict	bool	Set, frozenset





Type Conversions

The process of converting data of a particular type to another

01

Implicit Conversion

```
integer_number = 123
float_number = 1.23

new_number = integer_number + float_number

print("Value:",new_number)
print("Data Type:",type(new_number))
```

02

Explicit Conversion

```
num_string = '12'

# explicit type conversion
num_string = int(num_string)

print("Data type of num_string after Type
Casting:",type(num_string))
```

type(): it is a built-in function that returns the type of data.

int(): it receives input and converts it to int datatype.

Functions will be covered later inshallah!





Python I/O

```
graph TD; A[Python I/O] --> B[Python Output]; A --> C[Python Input];
```

Python Output

`print()` : function used to print in python.
However, it receives 5 parameters:

```
print(object= separator= end= file= flush= )
```

Example:

```
print('New Year', 2023, 'See you soon!', sep= '. ')
```

Python Input

`input()`: function used to take input from the user. It takes a string to prompt the user for input.

```
num = input('Enter a number: ')
```

```
print('You Entered:', num)
```





Python Operators



- **Arithmetic Operators:** + , - , * , / , % , **
- **Assignment Operators:** = , += , -= , *= , /= , %= , **=
- **Comparison Operators:** == , != , > , < , >= , <=
- **Logical Operators:** and , or , not
- **Bitwise Operators:** & , | , ~ , ^ , >> , <<
- **Special Operators:** is , is not , in , not in
 - is: return true if the operands are identical.
 - is not: return true if the operands are not identical.
 - in: return true if an item belongs to sequence.
 - not in: return false if an item not belongs to sequence.

```
a = 10
b = 5
a += b    # a = a + b
print(a)
```

```
a = 2
b = 4
print('Sum: ', a ** b)
```

```
a = 2
b = 3
print('a != b =', a != b)
```

```
print(True and False)
```

```
a = "hello"
print('h' not in a)
```

```
a = 2
b = 3
print(a is not b)
```



Python Flow Control



If statements

Decisions are required to be taken based on a condition in order to execute a piece of code.

If statement

Syntax:

```
if condition :  
    statement(s) to execute
```

Example:

```
num = 3  
if num > 0:  
    print(num, "is a positive number.")
```

If-else statement

Syntax:

```
if condition :  
    body of if  
else :  
    body of else
```

Example:

```
num = 3  
if num > 0:  
    print(num, "is a positive number.")  
else :  
    print(num, "is a negative number.")
```

If-elif-else statement

Syntax:

```
if condition :  
    body of if  
elif condition :  
    body of elif  
else :  
    body of else
```

Example:

```
if num > 0:  
    print("Positive number")  
elif num == 0:  
    print("Zero")  
else:  
    print("Negative number")
```

Nested if statement

Syntax:

```
if condition :  
    if condition :  
        body of inner if
```

Example:

```
num = float(input("Enter a number: "))  
if num >= 0:  
    if num == 0:  
        print("Zero")  
    else:  
        print("Positive number")  
else:  
    print("Negative number")
```



Python Flow Control

For Loop

For loop is used to iterate through a sequence of elements in a **list**, **tuples**, **directories**, and **strings**.

Syntax

```
for item in sequence:  
    body of for loop
```

Example:

```
# List of numbers  
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]  
sum = 0  
  
for val in numbers:  
    sum = sum+val  
  
print("The sum is", sum)
```

range(): it is a function that can be used with for loop to specify the range of iteration. We can define the start, end, and step size.

Example:

```
for i in range(start,end,step):  
    body of for loop
```

```
for i in range(2,20,3):  
    print(i)
```

While Loop

While loop is used to iterate through a block of statements as long as the condition is true.

Syntax

```
while condition:  
    body of for while
```

Example:

```
n = 10  
sum = 0  
i = 1  
  
while i <= n:  
    sum = sum + i  
    i = i+1 # update counter  
  
print("The sum is", sum)
```

Note: we can use an optional **else** statement to be executed when the condition in the **while loop** becomes **false**. Similarly, **for loop** iteration once finish, we can use an optional **else** clause.



Python break-continue

Python break statement

The break statement will terminate the loop once executed.

Example:


```
for val in "string":  
    if val == "i":  
        break  
    print(val)  
  
print("The end")
```

Python continue statement

The continue statement it will skip the remaining code in a current iteration and start a new iteration.

Example:

```
for val in "string":  
    if val == "i":  
        continue  
    print(val)  
  
print("The end")
```



Python Functions

A function is a set of related statements that will be executed when it is called.

■ The function accepts **arguments**
Or it could be a **non-argument** function.

■ The function can **return** a value
Or **non-returned type**.

■ The function can be a **user-defined** or **built-in** function.

Syntax:

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

Example of non-returned type function:

```
def greet(name):  
    print("Hello, " + name + ". Good morning!")
```

↪ This function named “**greet**” it will print the user a greeting message.

How to call the function? Simply by the name of the function

```
def greet(name):  
    print("Hello, " + name + ". Good morning!")  
  
greet("AI and Robot Club") # call
```

Example of returned type function:

```
def absolute_value(num):  
    if num >= 0:  
        return num  
    else:  
        return -num  
  
print(absolute_value(2))
```



Return statement: used to exit from a function and assign the returned value to a variable(maybe list, tuple), or simply print it.

Python Functions

Built-in functions: is a function written in Python Module we can call it implicitly or explicitly by importing its module. For example:

Built-in function

Implicit call such as: `print()` and `input()`

Import math module to use `sin()` function. (this will be covered in Python Modules!)

The function accepts arguments, or it could be a non-argument function.

The function can have default arguments:

```
def add_numbers( a = 7,
b = 8):
    sum = a + b
    print('Sum:', sum)

add_numbers(2, 3)

add_numbers(a = 2)
```

The function can have Keyword arguments:

```
def display_info(first_name, last_name):
    print('First Name:', first_name)
    print('Last Name:', last_name)

display_info(last_name = 'Cartman', first_name = 'Eric')
```

With an arbitrary number of arguments:

```
def find_sum(*numbers):
    result = 0

    for num in numbers:
        result = result + num

    print("Sum = ", result)
find_sum(1, 2, 3)
find_sum(4, 9)
```

Python Modules

A module is a file containing code to perform a specific task. It may contain variables, constants, classes, and functions. Also, we can create our own modules in python.

How to create a module?

1. First, let's name a file called "addition.py", then build a function to find the sum of two integers.

Addition.py

```
def add(a, b):  
    result = a + b  
    return result
```

How to use the module?

2. Import the module that you have created in main.py
By writing the "import" keyword with the name of the module.

```
import addition
```

3. Call the method "add" by writing: **addition.add(x,y)** , which x and y can be any two numbers.

```
print(addition.add(2,3)) # return 5
```

Python contains over 200 built-in modules, and we can import them the same way we did previously.

Let call **pi** value from **math** module:

```
import math
```

```
# use math.pi to get value of pi  
print("The value of pi is", math.pi)
```

Python Modules

■ We can rename the module

For example, let take the `math` module, we can rename it to `m`:

```
import math as m
print(m.pi)
```

■ Import all names

We can import all names(the definitions) inside the module except the private definitions that start with an underscore.

```
from math import *
print("The value of pi is", pi)
```

■ Python from...import statement

We can use from...import statement when we want to import specific functions, variables, or particular things inside the module.

```
# import only pi from math module
from math import pi
print(pi)
```

We used `*` in order to call all available definitions except the private ones. See how we can simply write “`pi`” without the name of the module.

Python Packages

As our application program becomes larger, and the number of modules may increase, we can organize our modules to be saved in a package. Different modules are saved in different packages.

How can we import a module within a package?

2 ways to import module play:

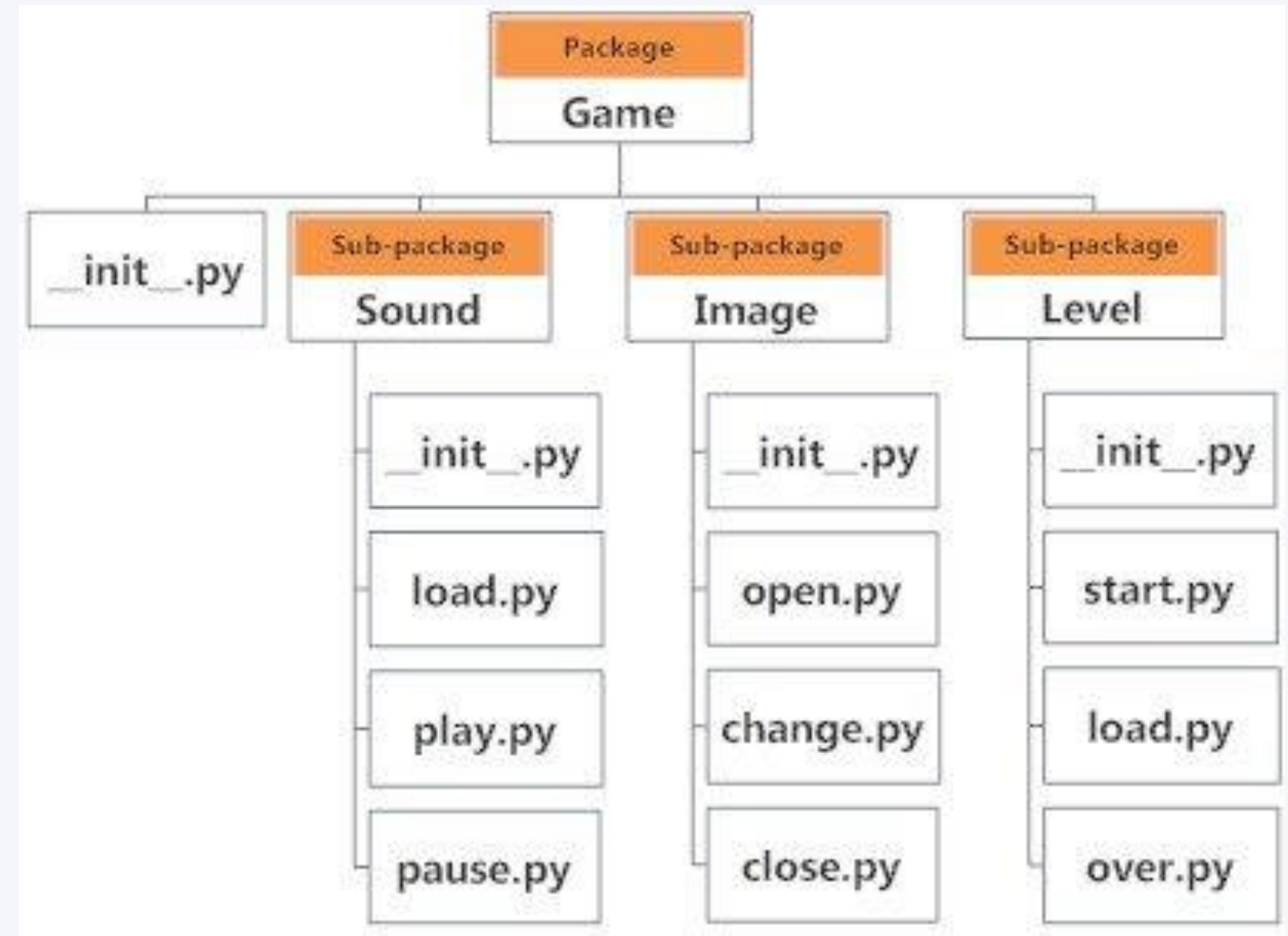
```
from Game.Sound import Play
import Game.Sound.Play
```

calling a function within the **play** module
named **move()**:

```
import Game.Sound.Play
Game.Sound.Play.move(3)
```

```
from Game.Sound import Play
Play.move(3)
```

```
from Game.Sound.Play import move
move(3)
```



Python Datatypes

Python List

Lists are used to store multiple elements and work with them at the same time

Syntax: `my_list1 = [1,2,3]`

- We can save mixed datatypes inside the list.

```
my_list2 = [1,"CCSIT",3.5]
```

- The list can be nested lists.

```
my_list3 = [1,[2,3],4]
```

- Access the element inside the list through the index.

```
Print(my_list3[1][0]) # output is 2
```

- Negative indexes allowed in python. It refers to the last elements. -1 for the last element, -2 for the second last element, and so on. For example:

```
Print(my_list1[-1])# output is 3
```

- The list slicing by taking a partition from list:

```
My_list[start index : end index -1]
```

- There is a list of function we can use to make a change on lists:
 - `append()`: add to the end of the list.
 - `extend()`: add all elements to another list.
 - `insert()`: insert at the specified index.
 - `remove()`: remove an item from list.
 - `clear()`: remove all list items.

Python Datatypes

Python Tuple Similar to list. However, we can not change the elements once assigned.

Syntax: `my_tuple = (1,2,3)`

- We can save mixed datatypes inside the tuple.

```
my_tuple2 = (1,"CCSIT",3.5)
```

- The tuple can be nested tuple.

```
my_tuple3 = (1,[2,3],4,(5,6,7),8)
```

- Tuple packing(): creating tuple without ()

```
my_tuple2 = 1,"CCSIT",3.5
```

- Tuple unpacking:

```
a , b , c = my_tuple2  
print(a) # 1
```

- Access a specific element through indexing, whether positive or negative, the same as lists.

```
Print(my_tuple2[0])# output is 1
```

- The list slicing by taking a partition from list:

```
My_tuple2[start index : end index -1]
```

- In order to delete a tuple:

```
del my_tuple2
```

- **count()** and **index()** functions will return the index of the item within the tuple.

Python Datatypes

Python Sets

A collection of unordered items, no duplicate items, and sets must be immutable.

Syntax: `my_set = { 1 , 2 , 3 }`

- We can save mixed datatypes inside the set.

```
my_set1 = { 1 , "CCSIT" , 3 }
```

- Update the set using **add** and **update** method:

```
my_set = {1, 3}

my_set.add(2)
print(my_set)

my_set.update([2, 3, 4])
print(my_set)

my_set.update([4, 5], {1, 6, 8})
print(my_set)
```

- We can remove an item using the **discard()** method, if the item is not present, will leave the list as it is.
- We can remove also using the **remove()** function, however, it will generate an error if the item not present.

```
my_set = {1, 3, 4, 5, 6}
my_set.discard(4)
print(my_set)

my_set.remove(6)
print(my_set)
```

- We can perform the other operations on sets:
insertion – difference – union – symmetric

Python Datatypes

Python Directories

A collection of ordered items, each item with key/value pair.

Syntax: `my_dict = { 1 : "CCSIT", 2 : "GDSC", 3 : "AI and Robot Club" }`

- access elements in directories using indexing, however, there is an error incase not found:

```
print(my_dict[2]) # GDSC
```

- Or using `get()` method, but here there is no error if the element not found.

```
print(my_dict.get(2)) # GDSC
```

- There is a list of functions we can use to make a change on directories:
 - **clear()**: remove all directory items.
 - **popitem()**: remove an item arbitrary
 - **Update()**: it is used to update an existing item, and overwrite the previous values.



Reference



Introduction to

Python

Team

- **Zeina Alabido**
 - **Reem Alshami**
 - **Raghad Aljiban**
 - **Aseel Alqahtani**
 - **Haya Almossaeed**
 - **Radwan Albahrani**
- 

Thank You

