Python Basics

Eng : Hesham Mohamed

What is Programming ?

- Programming is the process of giving machines a set of instructions that describe how a program should be carried out. Programmers will spend their whole careers learning a variety of programming languages and tools so they can effectively build computer programs.

What is Python ?

- **Python** is an object-oriented programming language created by Guido Rossum in 1989. It is ideally designed for rapid prototyping of complex applications. It has interfaces to many OS system calls and libraries and is extensible to C or C++. Many large companies use the Python programming language, including NASA, Google, YouTube, BitTorrent, etc.

Why Python ?

- Simple & Easy To Learn
- Blessed with large community
- Free and Open source
- Portable & Extensible
- You Can Do With Python Web & Mobile Applications , Data Science , Ai , Machine Learning && more

Python Is Simple !

Java

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world");
    }
}
```

Python

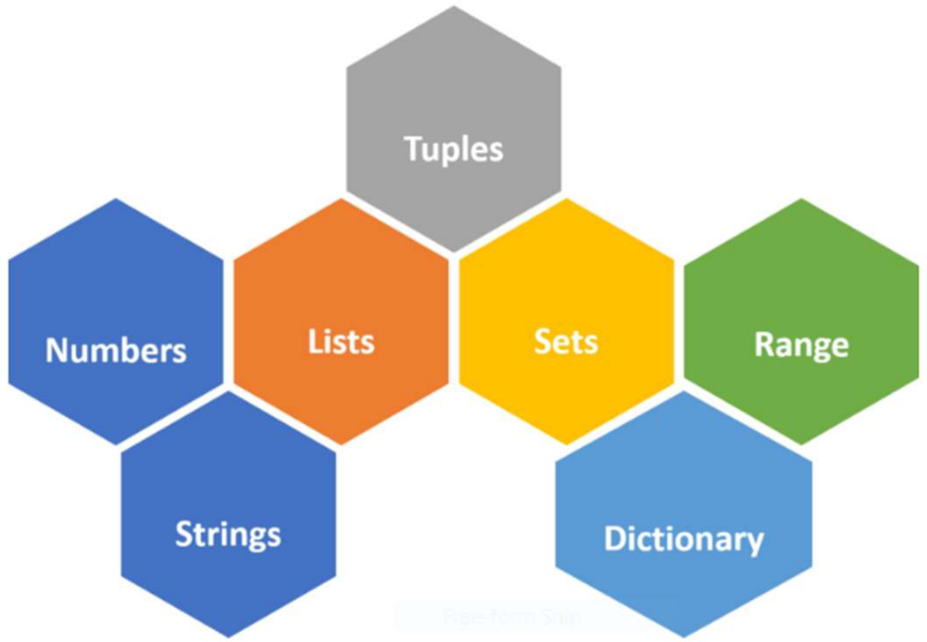
```python
print("Hello, world")
```

It’s that **SIMPLE**!

Variables

- a variable is a memory location where you store a value. The value that you have stored may change in the future according to the specifications.
- **There are a certain rules that we have to keep in mind while declaring a variable:**
- The variable name cannot start with a number. It can only start with a character or an underscore.
- Variables in python are case sensitive.
- They can only contain alpha-numeric characters and underscores.
- No special characters are allowed.

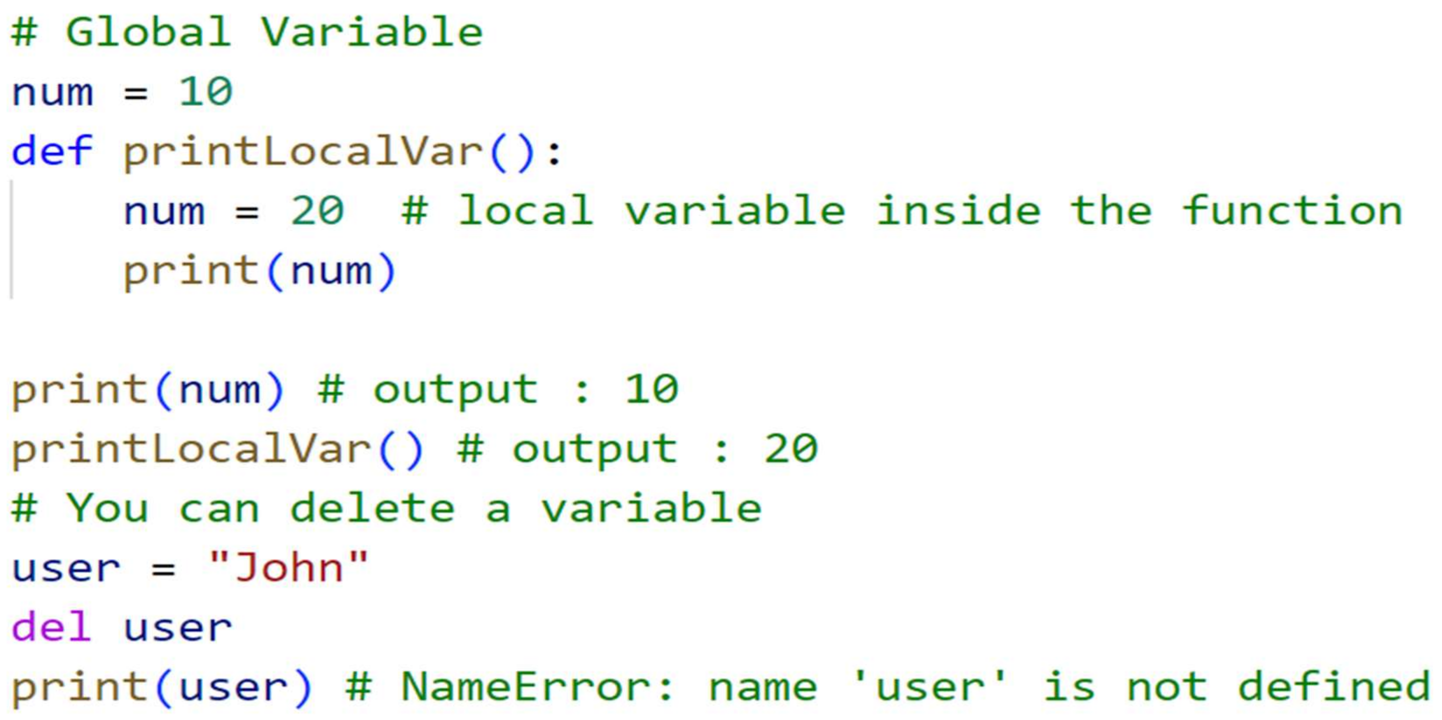
Data Types



Variables & Data Types

```
# This is Python Comment
# Declaring a variable and initializing it
user = "John"
# Check The Data Type of user
print(type(user)) # <str>
''' You can re-declare Python variables even
after you have declared once.'''
user = "Jack"
print(user) # output : jack
```

Python Variable Types: Local & Global



```python
# Global Variable
num = 10
def printLocalVar():
    num = 20  # local variable inside the function
    print(num)

print(num) # output : 10
printLocalVar() # output : 20
# You can delete a variable
user = "John"
del user
print(user) # NameError: name 'user' is not defined
```

Operators

- **Arithmetic Operators**
- Arithmetic Operators perform various arithmetic calculations like addition, subtraction, multiplication, division, %modulus, exponent, etc. There are various methods for arithmetic calculation in Python like you can use the eval function, declare variable & calculate, or call functions.
- **Comparison Operators**
- **Comparison Operators In Python** compares the values on either side of the operand and determines the relation between them. It is also referred to as relational operators. Various comparison operators in python are (==, != , <>, >,<=, etc.)

Operators

- **Assignment Operators**
- **Assignment Operators** in **Python** are used for assigning the value of the right operand to the left operand. Various assignment operators used in Python are (+=, – = , *=, /= , etc.).
- **Logical Operators or Bitwise Operators**
- Logical operators in Python are used for conditional statements are true or false. Logical operators in Python are AND, OR and NOT. For logical operators following condition are applied.
- For AND operator – It returns TRUE if both the operands (right side and left side) are true
- For OR operator- It returns TRUE if either of the operand (right side or left side) is true
- For NOT operator- returns TRUE if operand is false

Operators

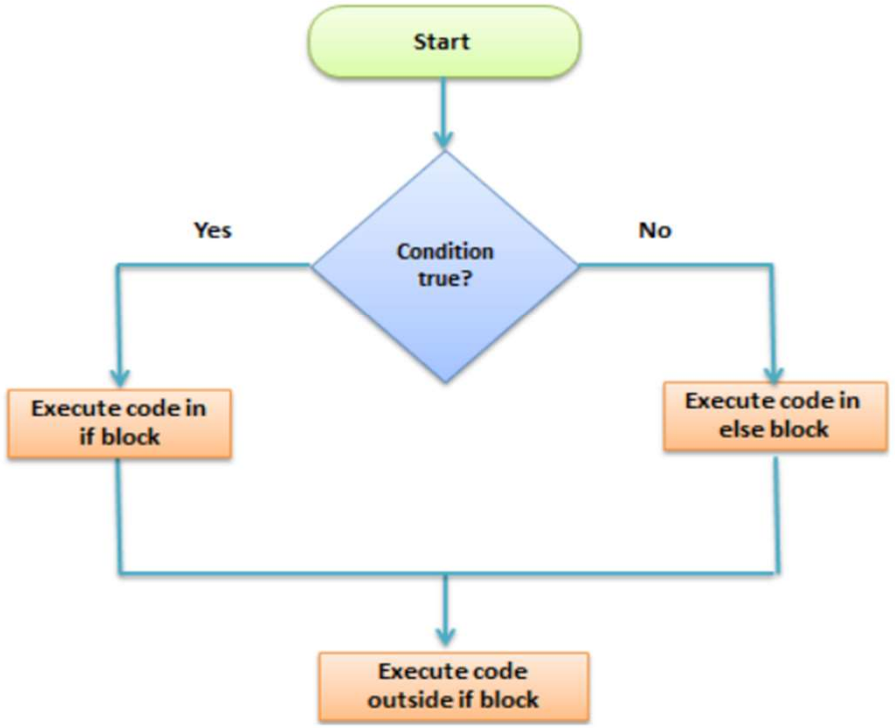
- **Membership Operators**
- These operators test for membership in a sequence such as lists, strings or tuples. There are two membership operators that are used in Python. (in, not in). It gives the result based on the variable present in specified sequence or string
- **Identity Operators**
- **Identity Operators in Python** are used to compare the memory location of two objects. The two identity operators used in Python are (is, is not).
- Operator is: It returns true if two variables point the same object and false otherwise
- Operator is not: It returns false if two variables point the same object and true otherwise

Operators

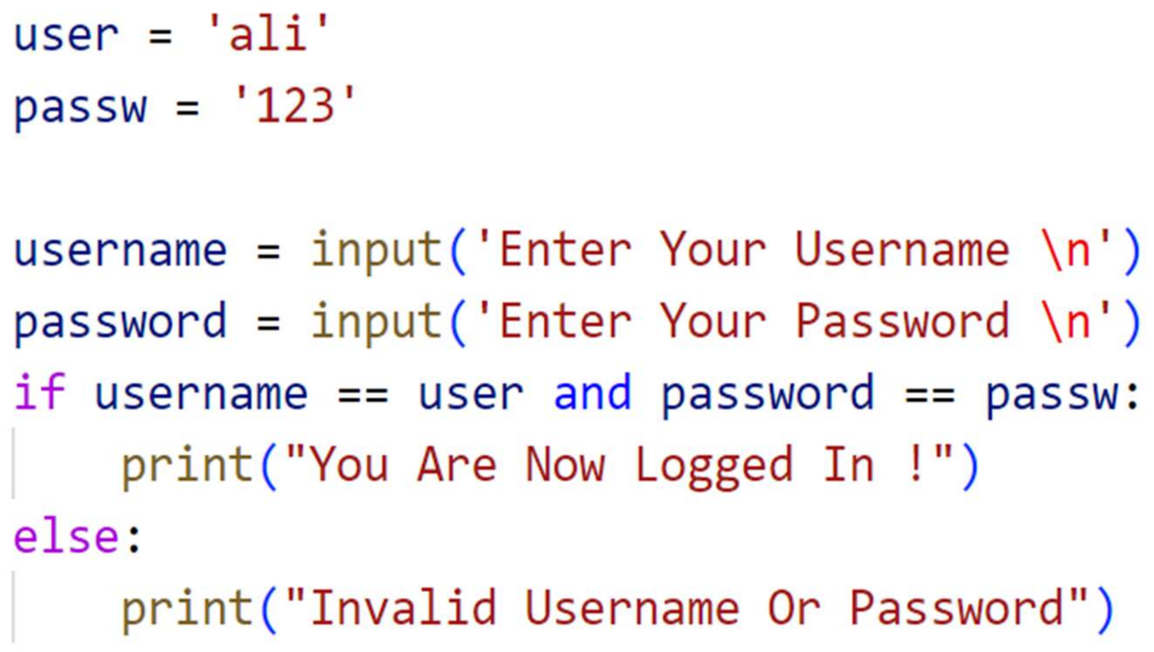
Operators (Decreasing order of precedence)	Meaning
**	Exponent
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, –	Addition, Subtraction
<= < > >=	Comparison operators
= %= /= //= -= += *= **=	Assignment Operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Conditions

Python if…else Flowchart

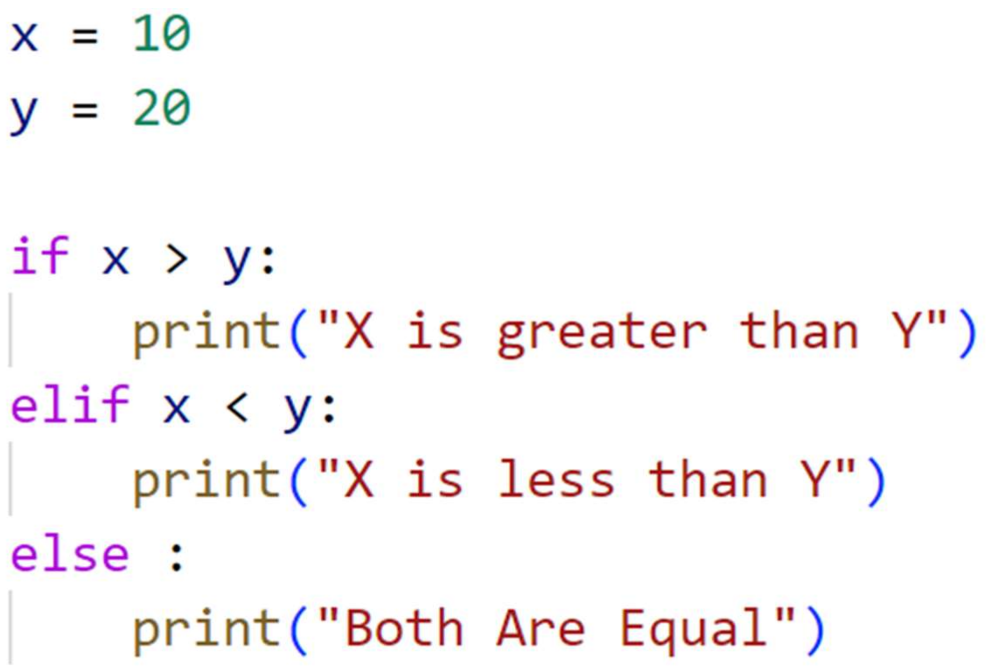


If .. Else Example



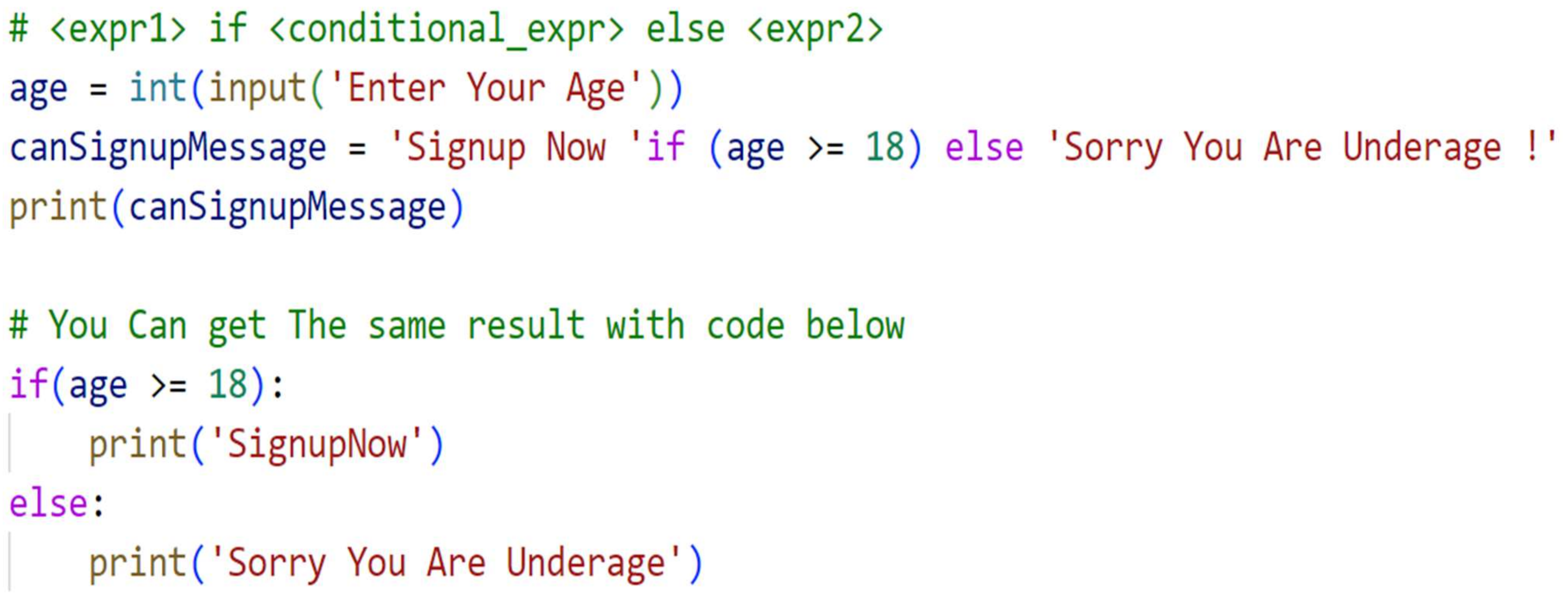
```
user = 'ali'
passw = '123'

username = input('Enter Your Username \n')
password = input('Enter Your Password \n')
if username == user and password == passw:
    print("You Are Now Logged In !")
else:
    print("Invalid Username Or Password")
```

If .. Elif .. Else Example



```python
x = 10
y = 20

if x > y:
    print("X is greater than Y")
elif x < y:
    print("X is less than Y")
else :
    print("Both Are Equal")
```

Conditional Expressions (Python’s Ternary Operator)



```python
# <expr1> if <conditional_expr> else <expr2>
age = int(input('Enter Your Age'))
canSignupMessage = 'Signup Now 'if (age >= 18) else 'Sorry You Are Underage !'
print(canSignupMessage)

# You Can get The same result with code below
if(age >= 18):
    print('SignupNow')
else:
    print('Sorry You Are Underage')
```

The Python pass Statement

- Occasionally, you may find that you want to write what is called a code stub: a placeholder for where you will eventually put a block of code that you haven’t implemented yet.
- In languages where token delimiters are used to define blocks, like the curly braces in Perl and C, empty delimiters can be used to define a code stub

- # This is not Python
- **if (x)**
- **{**
- **}**

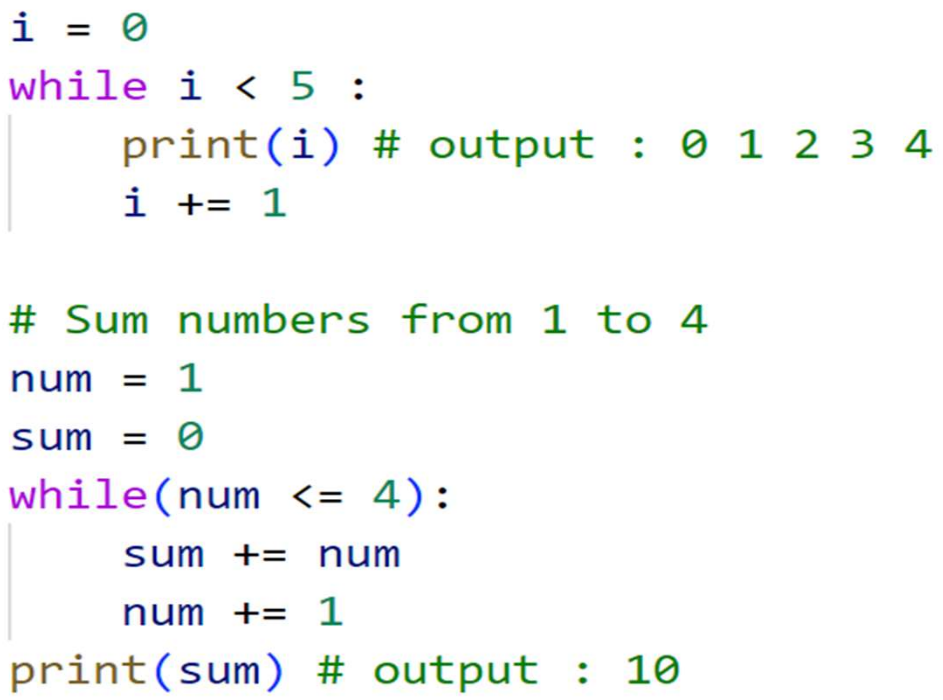
Loop

- **What is Loop?**
- Loops can execute a block of code number of times until a certain condition is met. Their usage is fairly common in programming. Unlike other programming language that have For Loop, while loop, dowhile, etc.
- **What is For Loop?**
- For loop is used to iterate over elements of a sequence. It is often used when you have a piece of code which you want to repeat “n” number of time.
- **What is While Loop?**
- While Loop is used to repeat a block of code. Instead of running the code block once, It executes the code block multiple times until a certain condition is met.

For Loop

```
for i in range(5):
    print(i) # output : 0 1 2 3 4

for i in range(0 , 5):
    print(i) # output : 0 1 2 3 4

for i in range(0 , 5 , 2):
    print(i) # output : 0 2 4
```

While Loop

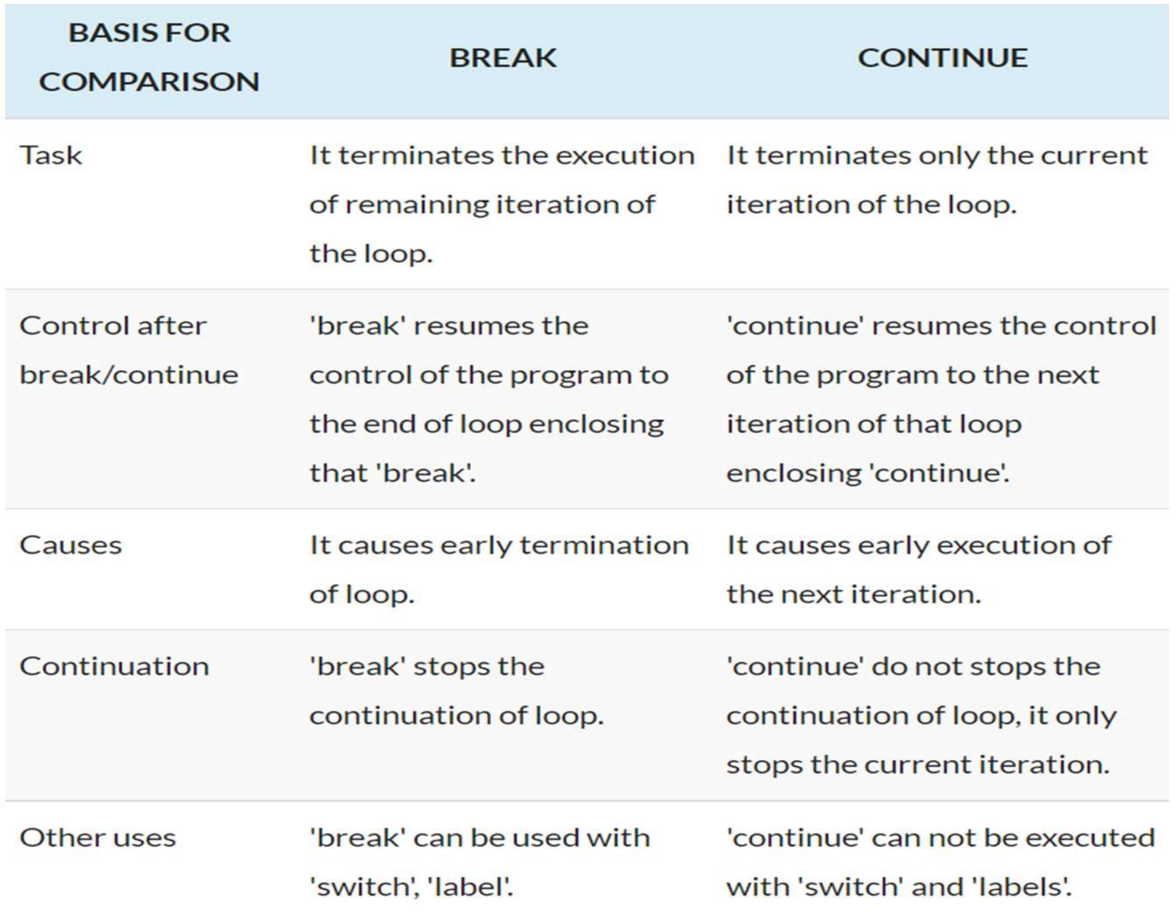


```python
i = 0
while i < 5 :
    print(i) # output : 0 1 2 3 4
    i += 1

# Sum numbers from 1 to 4
num = 1
sum = 0
while(num <= 4):
    sum += num
    num += 1
print(sum) # output : 10
```

Break vs Continue

- The main difference between break and continue is that break is used for immediate termination of loop. On the other hand, ‘continue’ terminate the current iteration and resumes the control to the next iteration of the loop.
- The break statement is primarily used as the exit statement, which helps in escaping from the current block or loop.

Break vs Continue



BASIS FOR COMPARISON	BREAK	CONTINUE
Task	It terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop.
Control after break/continue	'break' resumes the control of the program to the end of loop enclosing that 'break'.	'continue' resumes the control of the program to the next iteration of that loop enclosing 'continue'.
Causes	It causes early termination of loop.	It causes early execution of the next iteration.
Continuation	'break' stops the continuation of loop.	'continue' do not stops the continuation of loop, it only stops the current iteration.
Other uses	'break' can be used with 'switch', 'label'.	'continue' can not be executed with 'switch' and 'labels'.

Strings

```
user = 'John'
#this will make the letters to uppercase
user.upper()
#this will make the letters to lowercase
user.lower()
#this will replace the letter 'e' with 'E'
user.replace('j' , 'J')
#this will return the strings starting at index 1 until the index 4.
user[0:2]
text = 'Python is a fun programming language'
# split the text from space
print(text.split(' '))
# Output: ['Python', 'is', 'a', 'fun', 'programming', 'language']
```

Lists

- List is a collection data type in python. It is ordered and allows duplicate entries as well. Lists in python need not be homogeneous, which means it can contain different data types like integers, strings and other collection data types. It is mutable in nature and allows indexing to access the members in a list.

- List is like any other array that we declare in other programming languages. Lists in python are often used to implement stacks and queues. The lists are mutable in nature. Therefore, the values can be changed even after a list is declared.

Lists

```
# List of students
students = ['John' , 'Jack' , 'Mark']
students[0] # John
students[1] # Jack
students[2] # Mark
print(students.count('Jack')) # 1
# Append Luka to students
students.append('Luka')
print(students) #['John' , 'Jack' , 'Mark' , 'Luka']
# Empty the list
students.clear()
# Insert into students
students.insert(0 , 'Ali')
# Copy Students
copiedList = students.copy()
print(copiedList) #['Ali']
```

Lists

```
numbers = [1 , 2 , 5 , 3 , 4 , 5 , 4]
print(numbers.count(4)) # 2
# Get index of 2
print(numbers.index(2)) # 1
# Extend numbers
numbers.extend(range(6 , 11))
# Sort numbers
numbers.sort()
#Reverse numbers
numbers.reverse()
numbers[: :-1]
# Remove the last element
numbers.pop()
```

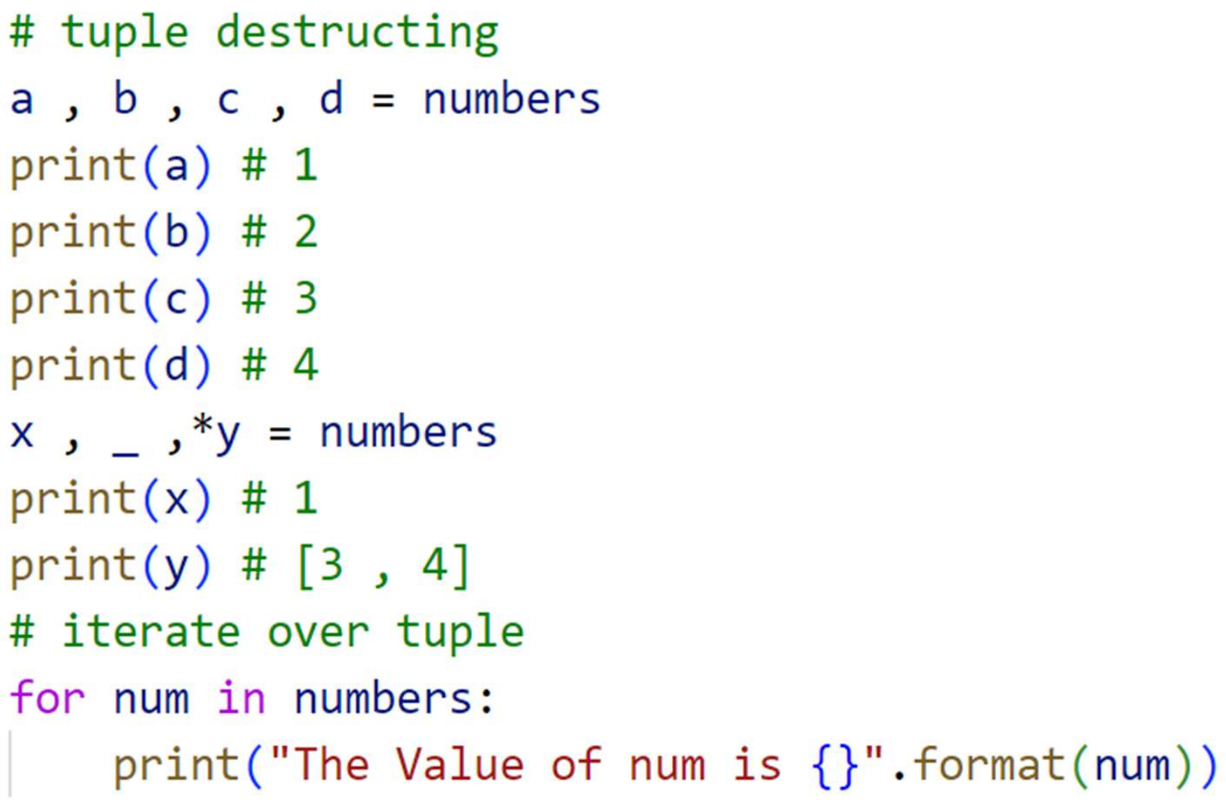
Lists

```
numbers = [1 , 2 , 5 , 3 , 4 , 5 , 4]
# Remove Any Value
numbers.remove(5)
# Slicing List
print(numbers[1 : 4]) #[2 , 3 , 4]
# Subsetting a list
anotherNumbers = [11 , 12 , 13 , 14 , numbers]
print(anotherNumbers[1]) # 12
print(anotherNumbers[5]) # IndexError: list index out of range
print(anotherNumbers[4][2]) # 3 => the third index in numbers
# Another Way to declaring list
users = list(('Ali' , 'Mohab' , 'Omar'))
print(users) # ['Ali' , 'Mohab' , 'Omar']
```

Tuples

```
# tuples are immutable
numbers = (1 , 2 , 3 , 4)
print(len(numbers)) # 1
print(numbers.count(2)) # 4
# Concatinating Tuples
anotherNumbers = (5 , 6 , 7 , 8)
print(numbers + anotherNumbers)
# get position from tuple
print(anotherNumbers.index(5)) # 0
```

Tuples

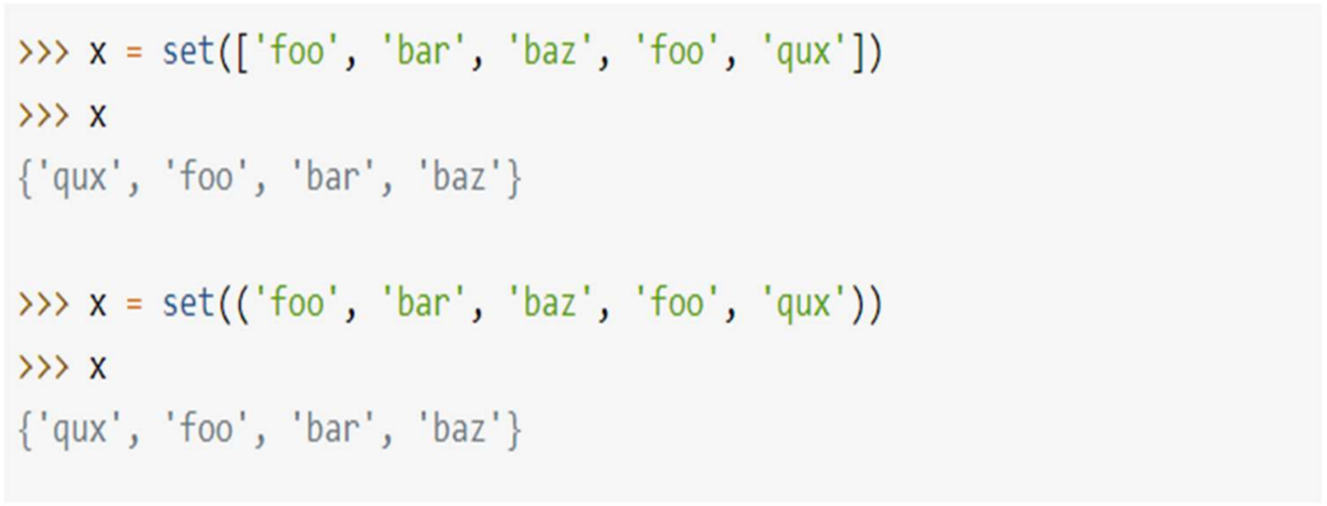


```python
# tuple destructing
a , b , c , d = numbers
print(a) # 1
print(b) # 2
print(c) # 3
print(d) # 4
x , _ ,*y = numbers
print(x) # 1
print(y) # [3 , 4]
# iterate over tuple
for num in numbers:
    print("The Value of num is {}".format(num))
```

Sets

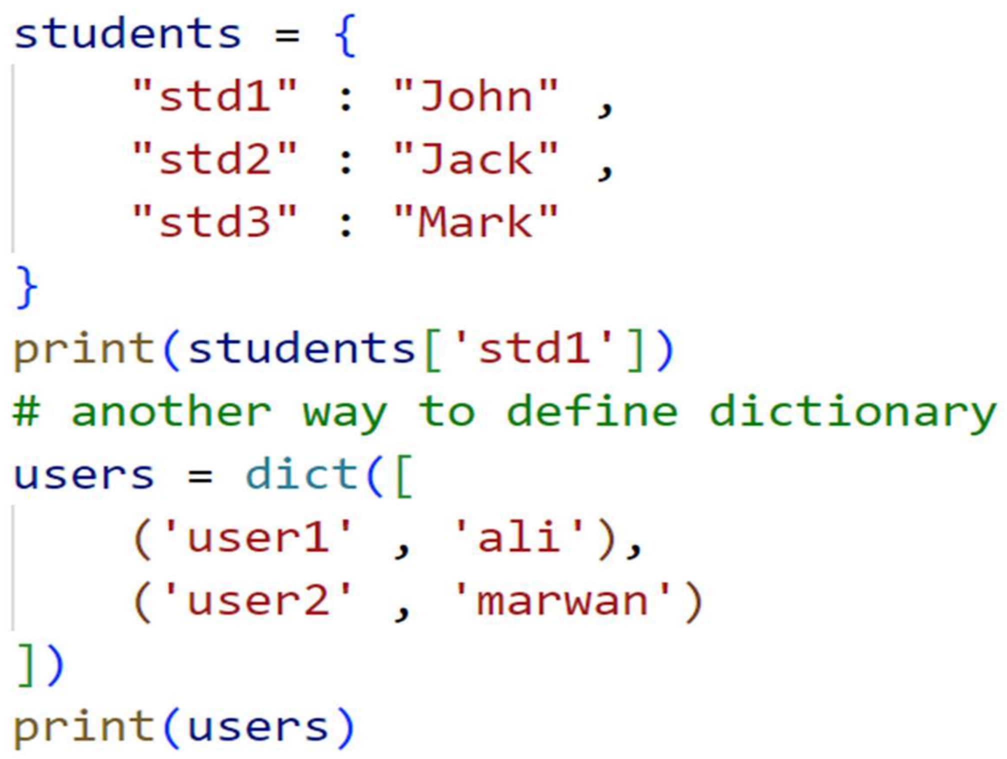
- ***Python's built-in set type has the following characteristics:***
- Sets are **unordered**.
- Set elements are **unique**. Duplicate elements are not allowed.
- A set itself may be modified, but the elements contained in the set must be of an **immutable type.**

Sets



```
>>> x = set(['foo', 'bar', 'baz', 'foo', 'qux'])
>>> x
{'qux', 'foo', 'bar', 'baz'}

>>> x = set(('foo', 'bar', 'baz', 'foo', 'qux'))
>>> x
{'qux', 'foo', 'bar', 'baz'}
```

Dictionary



```python
students = {
    "std1" : "John" ,
    "std2" : "Jack" ,
    "std3" : "Mark"
}
print(students['std1'])
# another way to define dictionary
users = dict([
    ('user1' , 'ali'),
    ('user2' , 'marwan')
])
print(users)
```

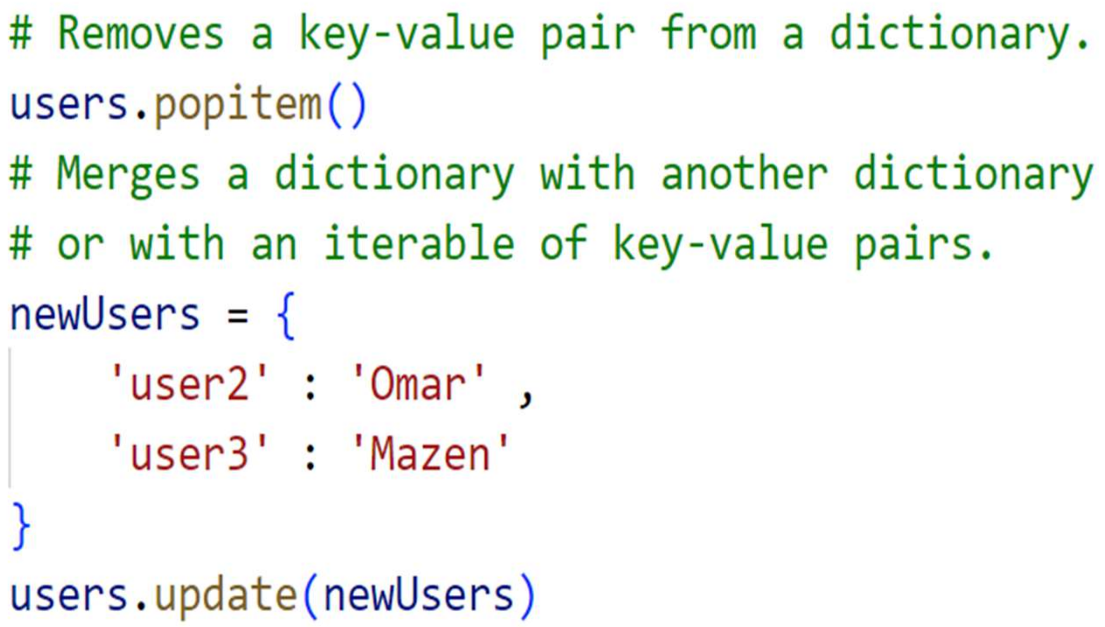
Dictionary

```python
# dictionary
students = {
    "std1" : "John" ,
    "std2" : "Jack" ,
    "std3" : "Mark"
}
print(students['std1'])
# another way to define dictionary
users = dict([
    ('user1' , 'ali'),
    ('user2' , 'marwan')
])
print(users)
```

Dictionary

```
# Dictionary methods
users.clear()
# Returns the value for a key if it exists in the dictionary.
#users.get(<key>[, <default>])
users.get('user1')
#Returns a list of key-value pairs in a dictionary.
users.items()
#Returns a list of keys in a dictionary.
users.keys()
#Returns a list of values in a dictionary.
users.values
#Removes a key from a dictionary, if it is present,
#  and returns its value.
users.pop('user1')
```

Dictionary



```python
# Removes a key-value pair from a dictionary.
users.popitem()
# Merges a dictionary with another dictionary
# or with an iterable of key-value pairs.
newUsers = {
    'user2' : 'Omar' ,
    'user3' : 'Mazen'
}
users.update(newUsers)
```

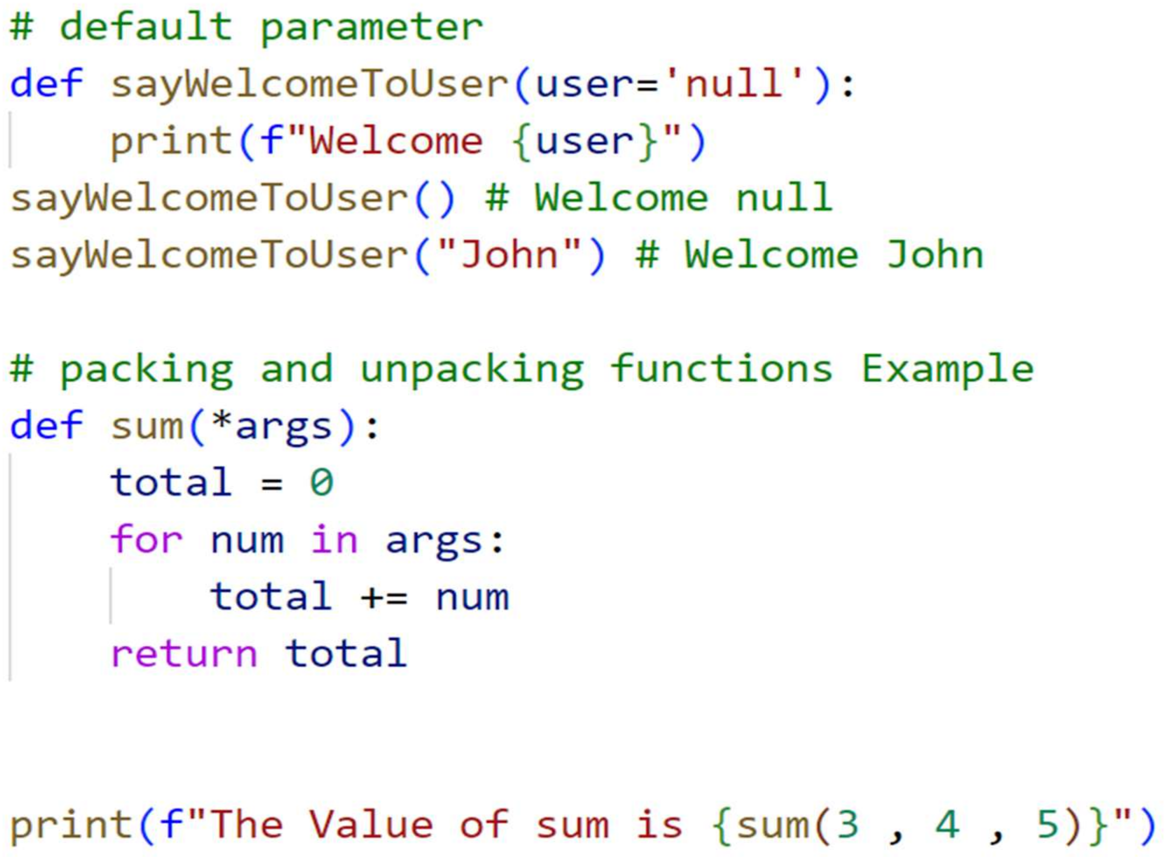
Functions

- a function is a group of related statements that performs 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.
- Furthermore, it avoids repetition and makes the code reusable.

Functions

```
#function without parameter
# this function is void
def sayWelcome():
    print('Welcome to python programming courses')
#calling
sayWelcome()

# function with parameter
# returned type <class int>
def add(num1 , num2):
    return num1 + num2

print(add(2 , 3)) # 5
print(type(add(2 , 3))) # <class 'int'>
```

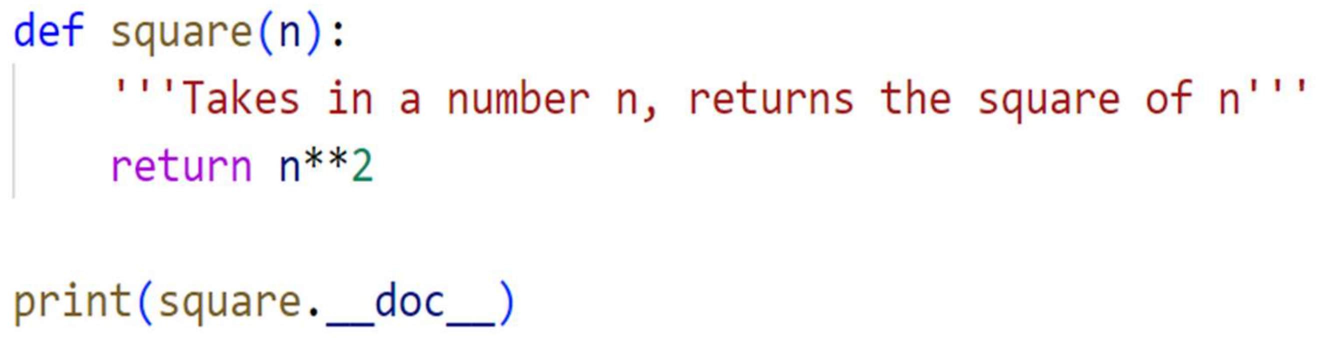
Functions



```python
# default parameter
def sayWelcomeToUser(user='null'):
    print(f"Welcome {user}")
sayWelcomeToUser() # Welcome null
sayWelcomeToUser("John") # Welcome John

# packing and unpacking functions Example
def sum(*args):
    total = 0
    for num in args:
        total += num
    return total


print(f"The Value of sum is {sum(3 , 4 , 5)}")
```

Docstrings

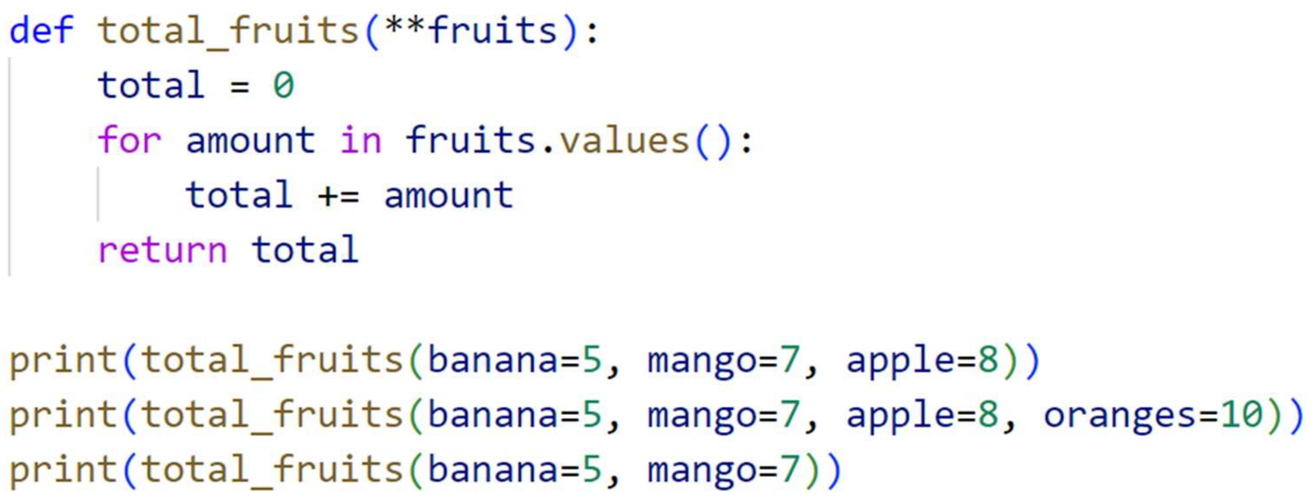
- Python docstrings are the string literals that appear right after the definition of a function, method, class, or module



```python
def square(n):
    '''Takes in a number n, returns the square of n'''
    return n**2

print(square.__doc__)
```

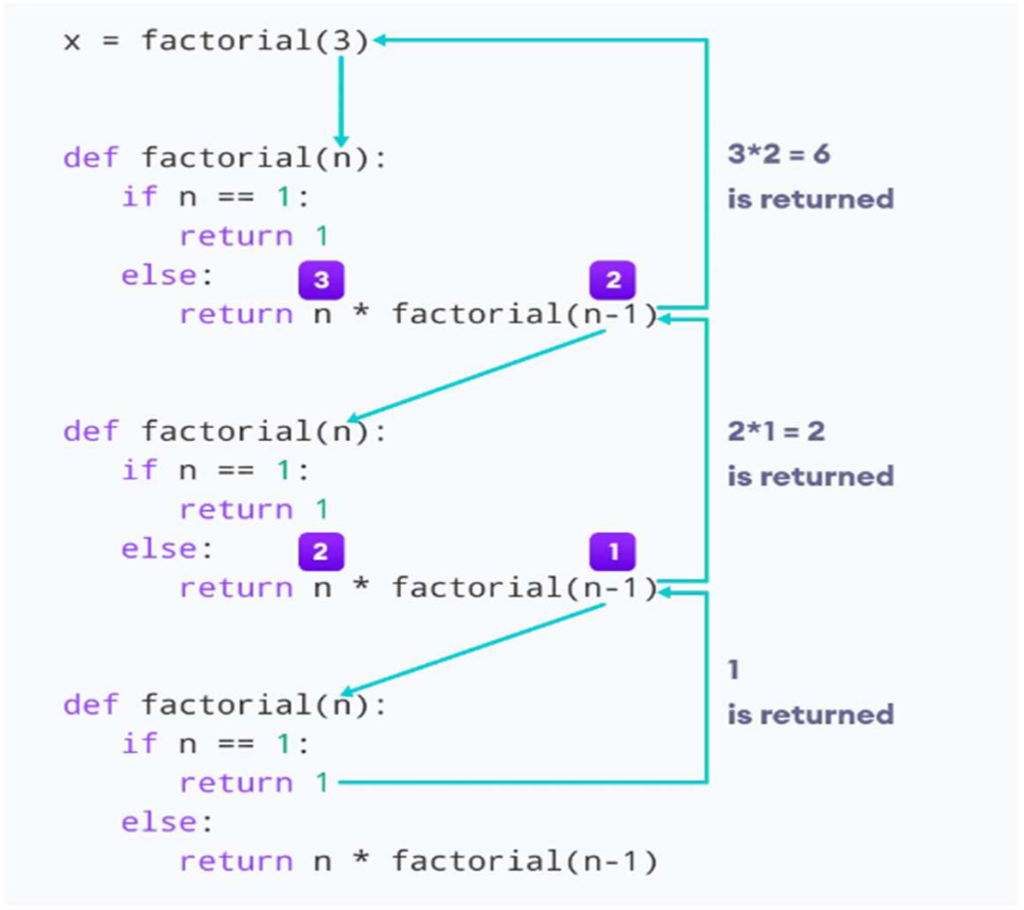
*args and **kwargs

- *args and **kwargs. These make a Python function flexible so it can accept a variable number of arguments and keyword arguments, respectively.



```python
def total_fruits(**fruits):
    total = 0
    for amount in fruits.values():
        total += amount
    return total

print(total_fruits(banana=5, mango=7, apple=8))
print(total_fruits(banana=5, mango=7, apple=8, oranges=10))
print(total_fruits(banana=5, mango=7))
```

Factorial Example

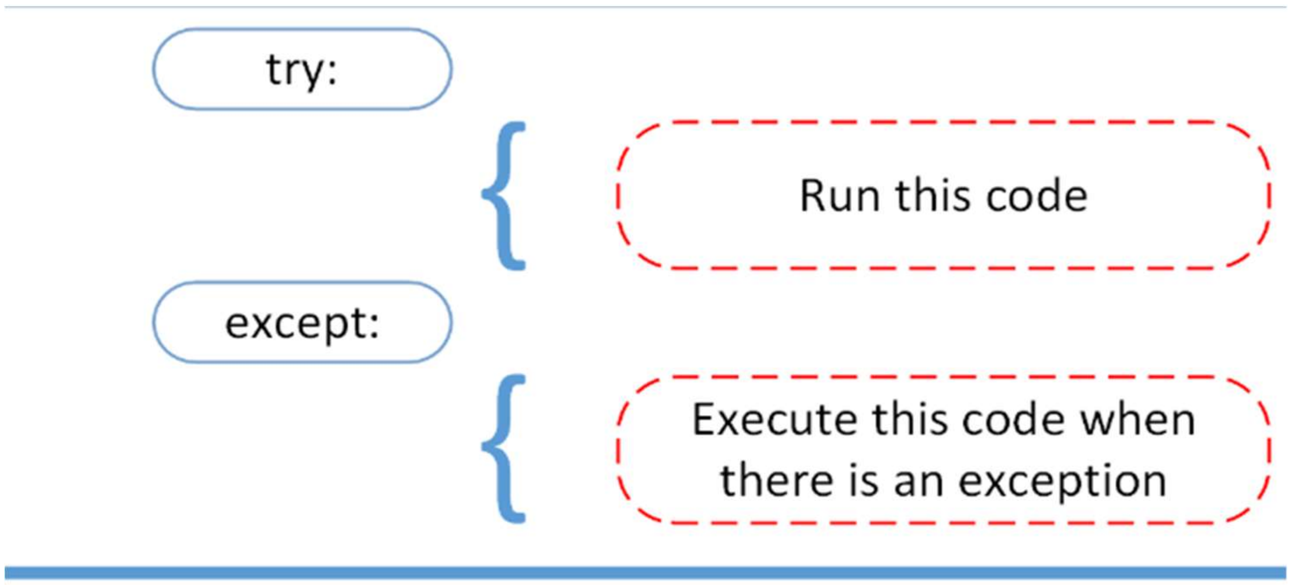


```
x = factorial(3)


def factorial(n):
   if n == 1:
      return 1
   else:
      return n * factorial(n-1)


def factorial(n):
   if n == 1:
      return 1
   else:
      return n * factorial(n-1)


def factorial(n):
   if n == 1:
      return 1
   else:
      return n * factorial(n-1)
```

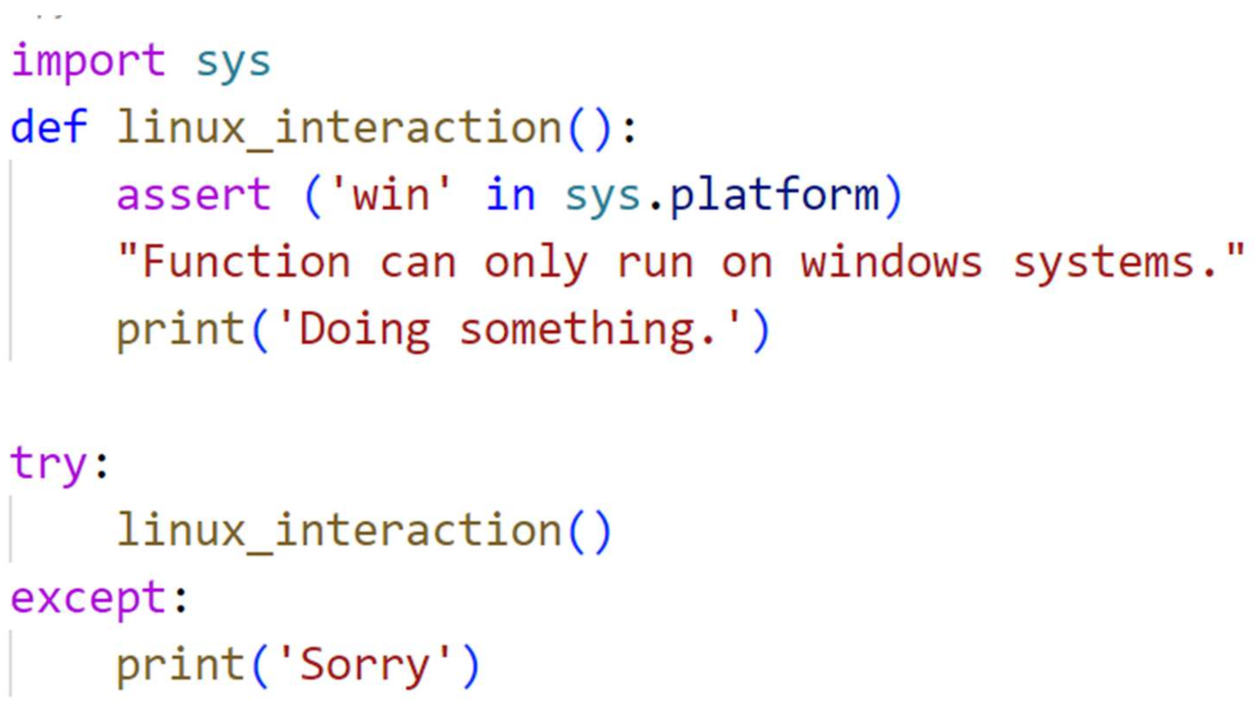
Recursion

- **Advantages of Recursion**
- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.
- **Disadvantages of Recursion**
- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

The try and except Block: Handling Exceptions

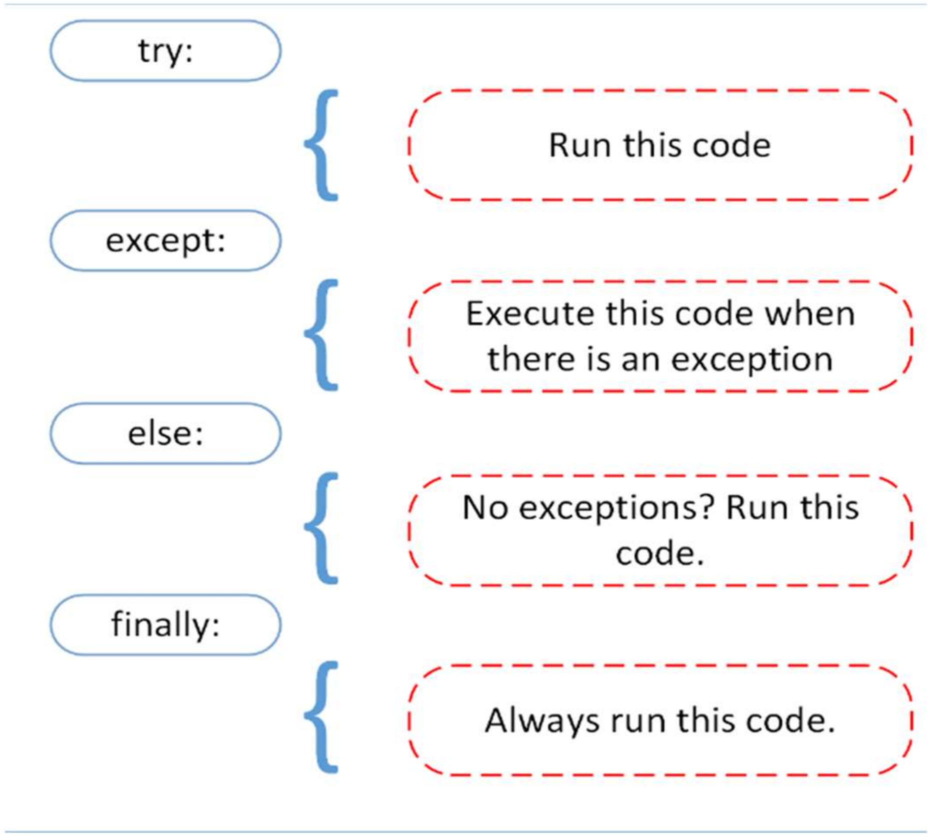


The try and except Block: Handling Exceptions



```
import sys
def linux_interaction():
    assert ('win' in sys.platform)
    "Function can only run on windows systems."
    print('Doing something.')

try:
    linux_interaction()
except:
    print('Sorry')
```

Handling Exceptions



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

Any Questions ?

Thank You ☺