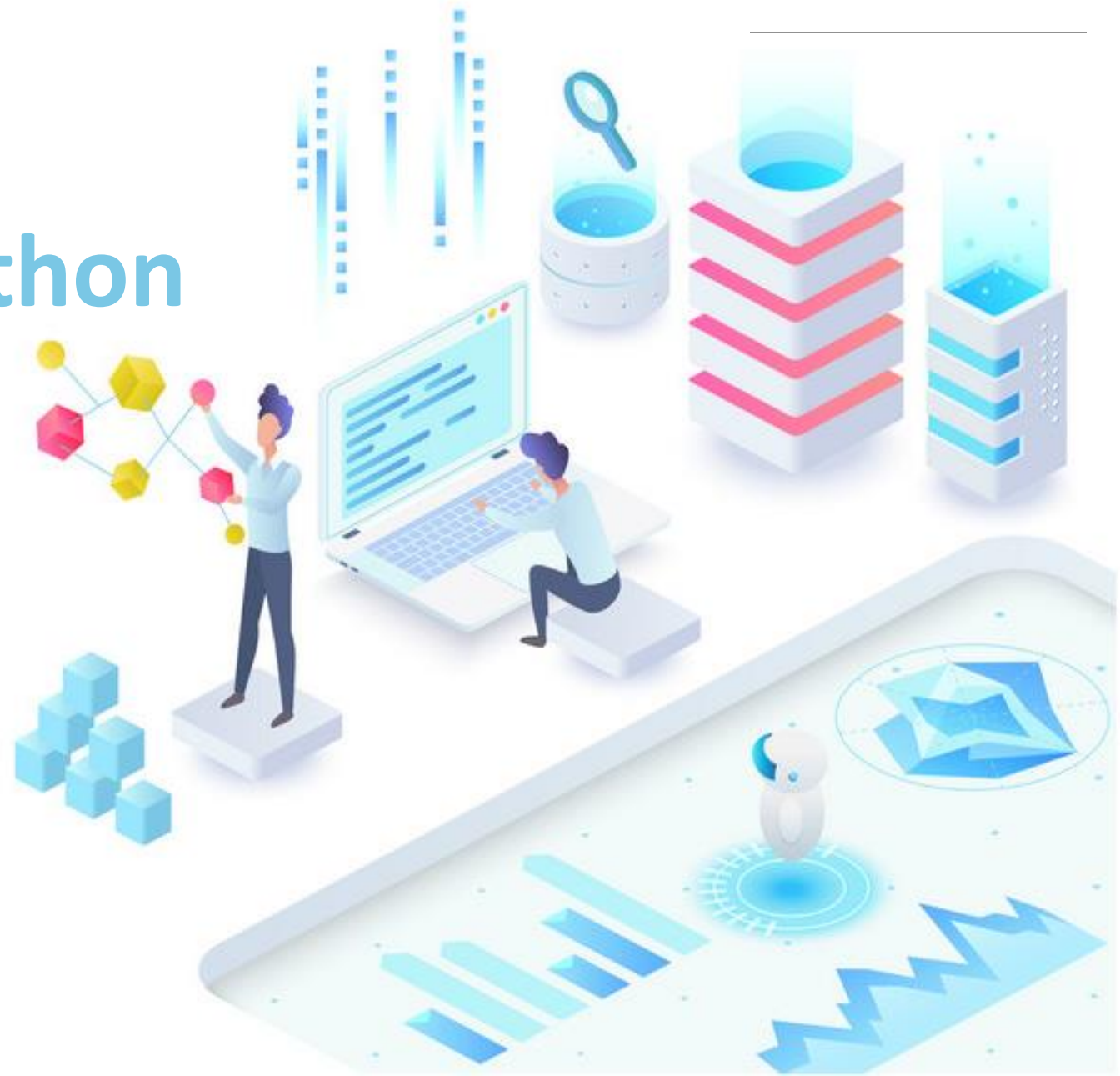
Markov ACADEMY

Data Science and Python

Eman Raslan



Python Basics

Data Types

Name	Type	Description	Example
Integers	Int	Whole numbers	3 300 200 -2
Floats	Float	Numbers with decimal point	2.3 4.0 -5.3
Strings	Str	Ordered sequence of characters	“hello” ‘Eman’ “2000”
Lists	List	Ordered sequence of objects	[10,”omar”,2.33]
Dictionaries	dict	Unordered Key:Value pairs	{“key1”:”value1”, “key2”:”value2”}
Tuples	Tup	Ordered immutable sequence of objects	(“hello”,4.4,200)
Sets	Set	Unordered collection of unique objects	{“a”,3000}
Booleans	bool	Logical vale indicating True or False	True False

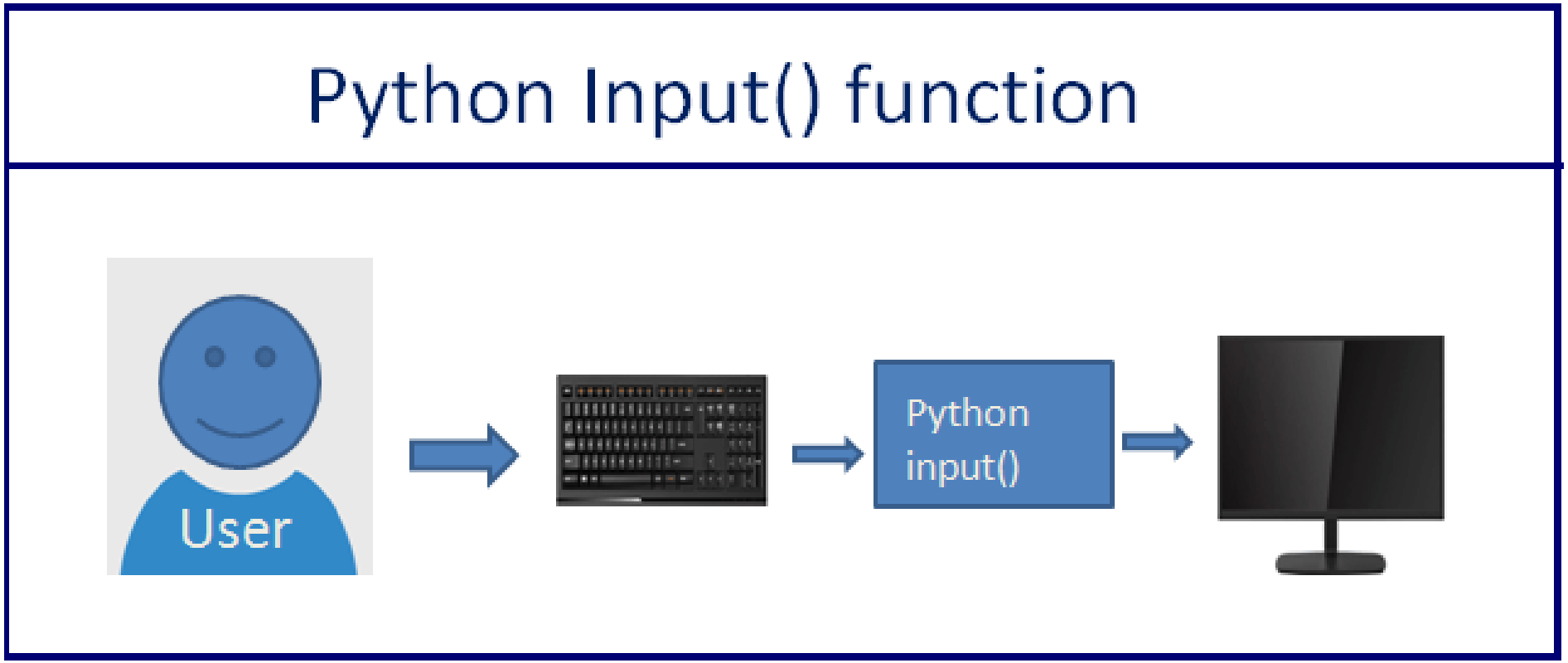
Arithmetic Operators

- `+` (Addition)
- `-` (Subtraction)
- `*` (Multiplication)
- `/` (Division)
- `//` Floor division)
- `%` (Modulus)
- `**` (Exponentiation)

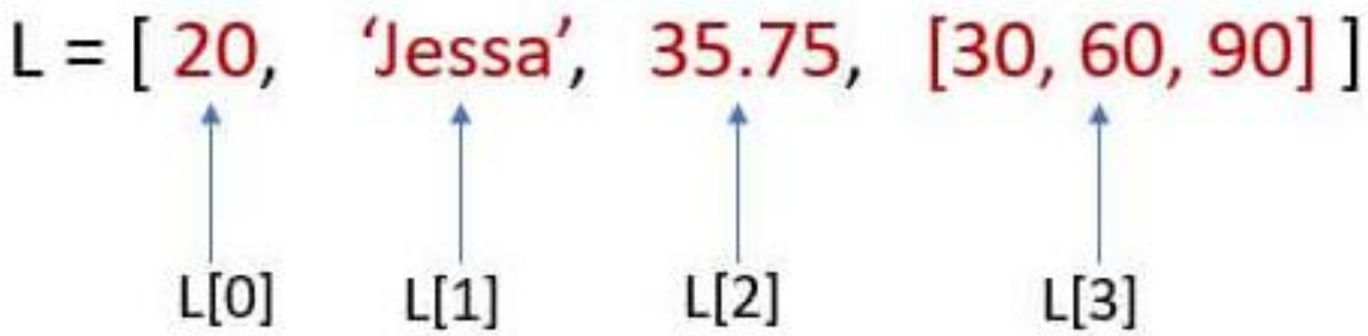
Assignment operators

Operator	Meaning	Equivalent
= (Assign)	a=5Assign 5 to variable a	a = 5
+= (Add and assign)	a+=5Add 5 to a and assign it as a new value to a	a = a+5
-= (Subtract and assign)	a-=5Subtract 5 from variable a and assign it as a new value to a	a = a-5
*= (Multiply and assign)	a*=5Multiply variable a by 5 and assign it as a new value to a	a = a*5
/= (Divide and assign)	a/=5Divide variable a by 5 and assign a new value to a	a = a/5
%= (Modulus and assign)	a%=5Performs modulus on two values and assigns it as a new value to a	a = a%5
**= (Exponentiation and assign)	a**=5Multiply a five times and assigns the result to a	a = a**5
//= (Floor-divide and assign)	a//=5Floor-divide a by 5 and assigns the result to a	a = a//5

Input



Lists



- ✓ **Ordered**: Maintain the order of the data insertion.
- ✓ **Changeable**: List is mutable and we can modify items.
- ✓ **Heterogeneous**: List can contain data of different types
- ✓ **Contains duplicate**: Allows duplicates data

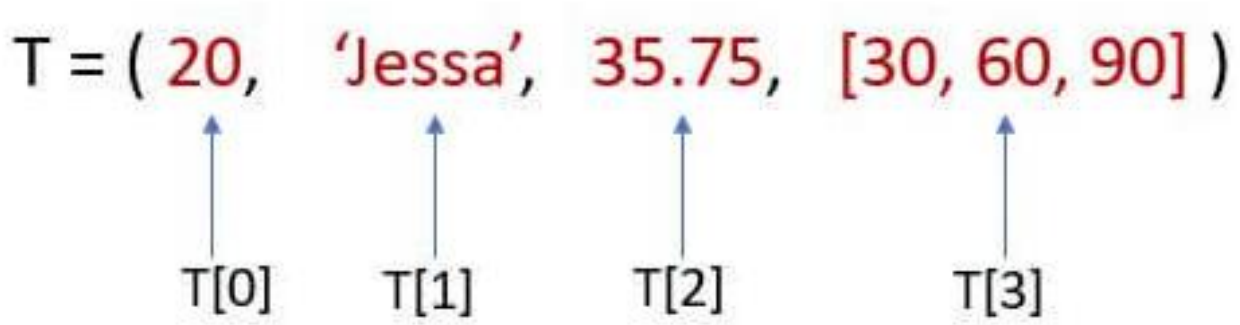
List Operations

Operation	Description
x in l1	Check if the list l1 contains item x.
x not in l2	Check if list l1 does not contain item x.
l1 + l2	Concatenate the lists l1 and l2. Creates a new list containing the items from l1 and l2.
l1 * 5	Repeat the list l1 5 times.
l1[i]	Get the item at index i. Example l1[2] is 30.
l1[i:j]	List slicing. Get the items from index i up to index j (excluding j) as a List. An example l1[0:2] is [10, 20]
l1[i:j:k]	List slicing with step. Returns a List with the items from index i up to index j taking every k-th item. An example l1[0:4:2] is [10, 30].
len(l1)	Returns a count of total items in a list.
l2.count(60)	Returns the number of times a particular item (60) appears in a list. The answer is 2.

List Operations

Operation	Description
l1.index(30)	Returns the index number of a particular item (30) in a list. The answer is 2.
l1.index(30, 2, 5)	Returns the index number of a particular item (30) in a list. But search Returns the item with maximum value from a list. The answer is 60 only from index number 2 to 5.
min(l1)	Returns the item with a minimum value from a list. The answer is 10.
max(l1)	Returns the item with maximum value from a list. The answer is 60.
l1.append(100)	Add item at the end of the list
l1.append([2, 5, 7])	Append the nested list at the end
l1[2] = 40	Modify the item present at index 2
l1.remove(40)	Removes the first occurrence of item 40 from the list.
pop(2)	Removes and returns the item at index 2 from the list.
l1.clear()	Make list empty
l3= l1.copy()	Copy l1 into l2

Tuples



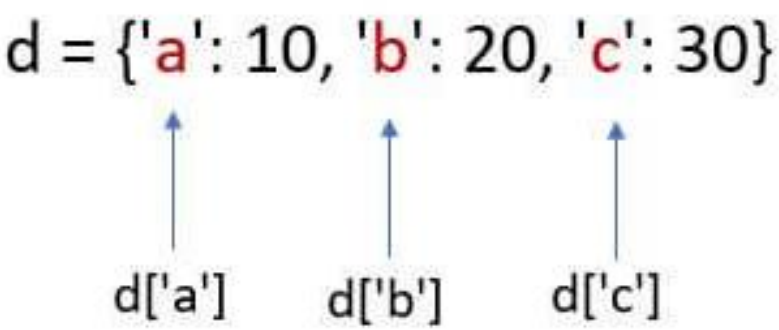
- ✓ **Ordered**: Maintain the order of the data insertion.
- ✓ **Unchangeable**: Tuples are immutable and we can't modify items.
- ✓ **Heterogeneous**: Tuples can contains data of types
- ✓ **Contains duplicate**: Allows duplicates data

tuples operations

Operation	Description
x in t1	Check if the tuple t1 contains the item x.
x not in t2	Check if the tuple t1 does not contain the item x.
t1 + t2	Concatenate the tuples t1 and t2. Creates a new tuple containing the items from t1 and t2.
t1 * 5	Repeat the tuple t1 5 times.
t1[i]	Get the item at the index i. Example, t1[2] is 30
t1[i:j]	Tuple slicing. Get the items from index i up to index j (excluding j) as a tuple. An example t1[0:2] is (10, 20)
t1[i:j:k]	Tuple slicing with step. Return a tuple with the items from index i up to index j taking every k-th item. An example t1[0:4:2] is (10, 30)
len(t1)	Returns a count of total items in a tuple
t2.count(60)	Returns the number of times a particular item (60) appears in a tuple. Answer is 2
t1.index(30)	Returns the index number of a particular item(30) in a tuple. Answer is 2
t1.index(40, 2, 5)	Returns the index number of a particular item(30) in a tuple. But search only from index number 2 to 5.
min(t1)	Returns the item with a minimum value from a tuple
max(t1)	Returns the item with maximum value from a tuple

Dictionary

Unordered collections of unique values stored in (Key-Value) pairs.



- ✓ **Unordered**: The items in dict are stored without any index value
- ✓ **Unique**: Keys in dictionaries should be Unique
- ✓ **Mutable**: We can add/Modify/Remove key-value after the creation

Dictionary Operations

Operations	Description
dict({'a': 10, 'b': 20})	Create a dictionary using a dict() constructor.
d2 = {}	Create an empty dictionary.
d1.get('a')	Retrieve value using the key name a.
d1.keys()	Returns a list of keys present in the dictionary.
d1.values()	Returns a list with all the values in the dictionary.
d1.items()	Returns a list of all the items in the dictionary with each key-value pair inside a tuple.
len(d1)	Returns number of items in a dictionary.
d1['d'] = 40	Update dictionary by adding a new key.
d1.update({'e': 50, 'f': 60})	Add multiple keys to the dictionary.
d1.setdefault('g', 70)	Set the default value if a key doesn’t exist.
d1['b'] = 100	Modify the values of the existing key.
d1.pop('b')	Remove the key b from the dictionary.
d1.popitem()	Remove any random item from a dictionary.
d1.clear()	Removes all items from the dictionary.
'key' in d1.keys()	Check if a key exists in a dictionary.
d1.update(d2)	Add all items of dictionary d2 into d1.
d3= {**d1, **d2}	Join two dictionaries.
d2 = d1.copy()	Copy dictionary d1 into d2.
max(d1)	Returns the key with the maximum value in the dictionary d1
min(d1)	Returns the key with the minimum value in the dictionary d1

Set

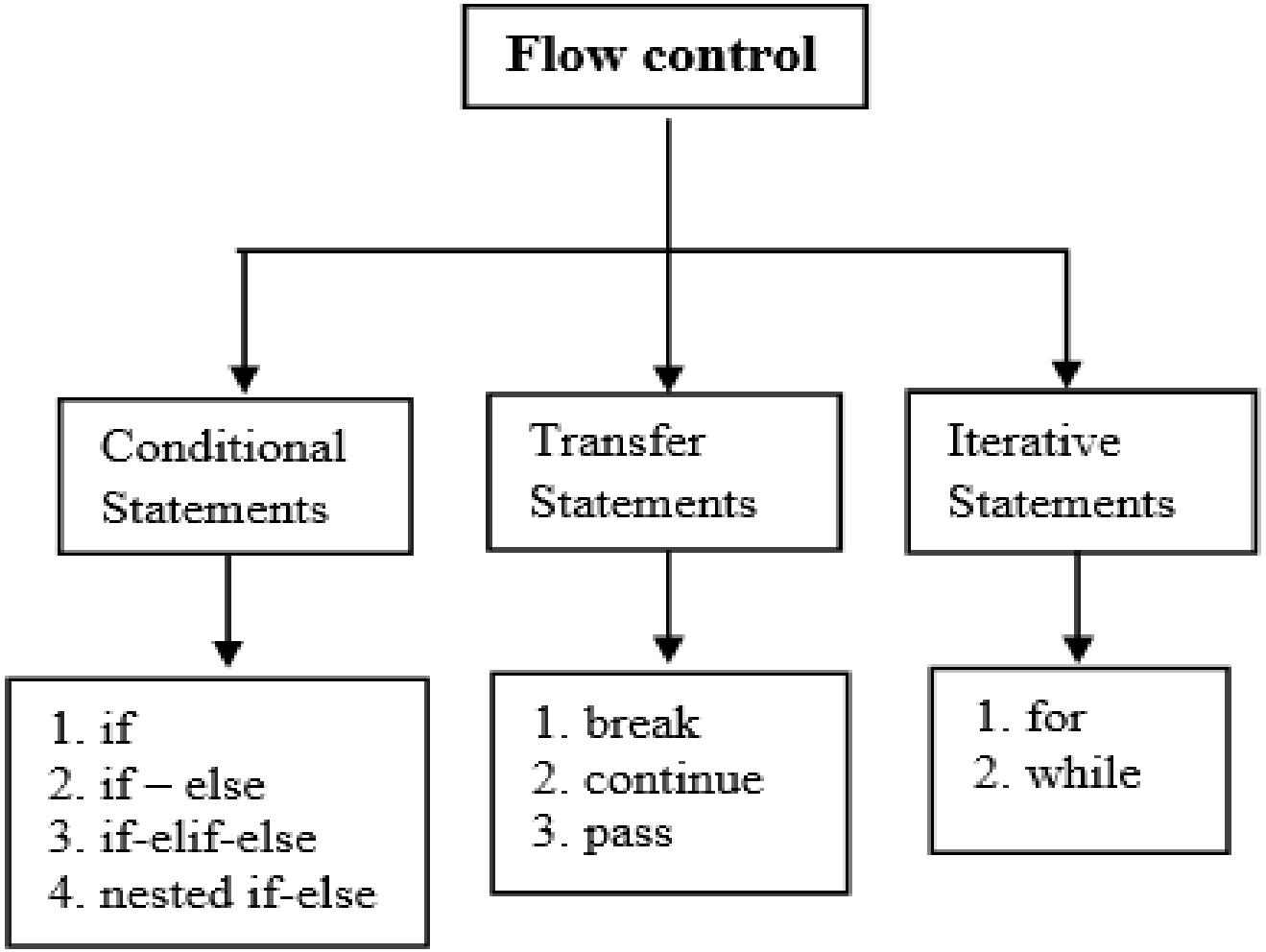
S = { 20, 'Jessa', 35.75}

- ✓ **Unordered**: Set doesn't maintain the order of the data insertion.
- ✓ **Unchangeable**: Set are immutable and we can't modify items.
- ✓ **Heterogeneous**: Set can contains data of all types
- ✓ **Unique**: Set doesn't allows duplicates items

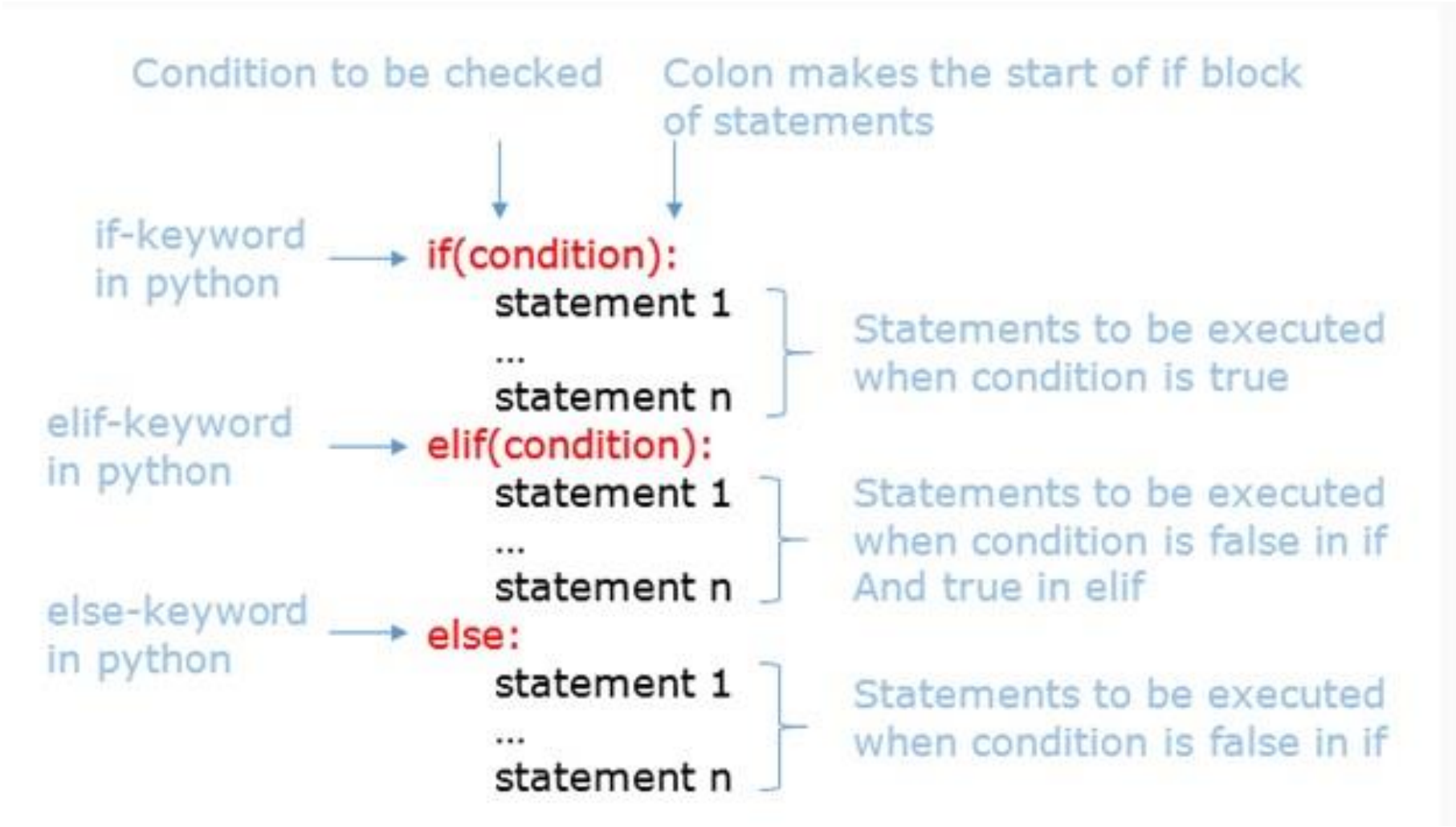
Set Operations

Operation	Equivalent	Result
`len(s)`		number of elements in set *s* (cardinality)
`x in s`		test *x* for membership in *s*
`x not in s`		test *x* for non-membership in *s*
`s.issubset(t)`	`s <= t`	test whether every element in *s* is in *t*
`s.issuperset(t)`	`s >= t`	test whether every element in *t* is in *s*
`s.union(t)`	`s | t`	new set with elements from both *s* and *t*
`s.intersection(t)`	`s & t`	new set with elements common to *s* and *t*
`s.difference(t)`	`s - t`	new set with elements in *s* but not in *t*
`s.symmetric_difference(t)`	`s ^ t`	new set with elements in either *s* or *t* but not both
`s.copy()`		new set with a shallow copy of *s*

Flow Control



if, elif, and else Statements



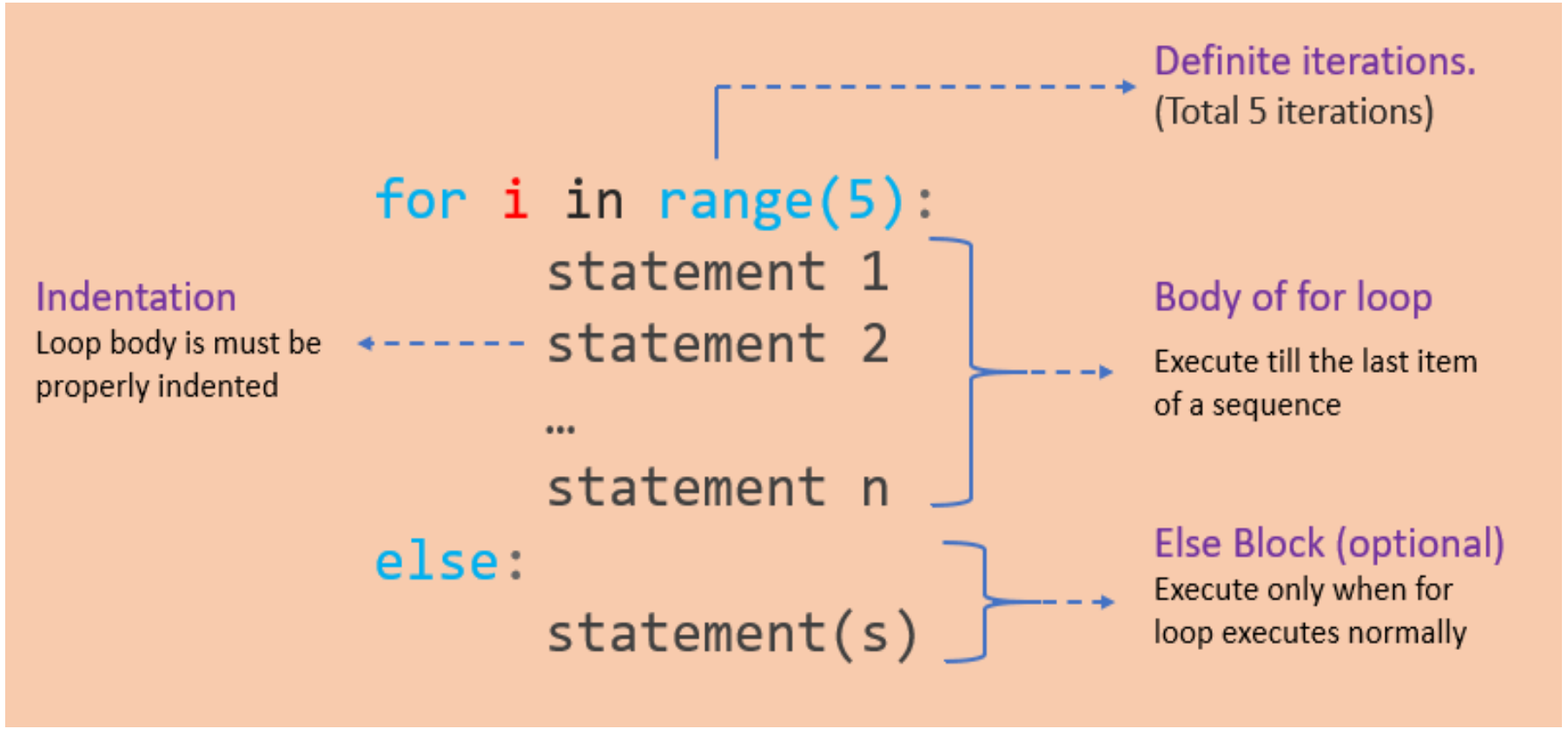
Relational (comparison) operators

Operator	Description	Example
> (Greater than)	It returns True if the left operand is greater than the right	x > y result is True
< (Less than)	It returns True if the left operand is less than the right	x < y result is False
== (Equal to)	It returns True if both operands are equal	x == y result is False
!= (Not equal to)	It returns True if both operands are equal	x != y result is True
>= (Greater than or equal to)	It returns True if the left operand is greater than or equal to the right	x >= y result is True
<= (Less than or equal to)	It returns True if the left operand is less than or equal to the right	x <= y result is False

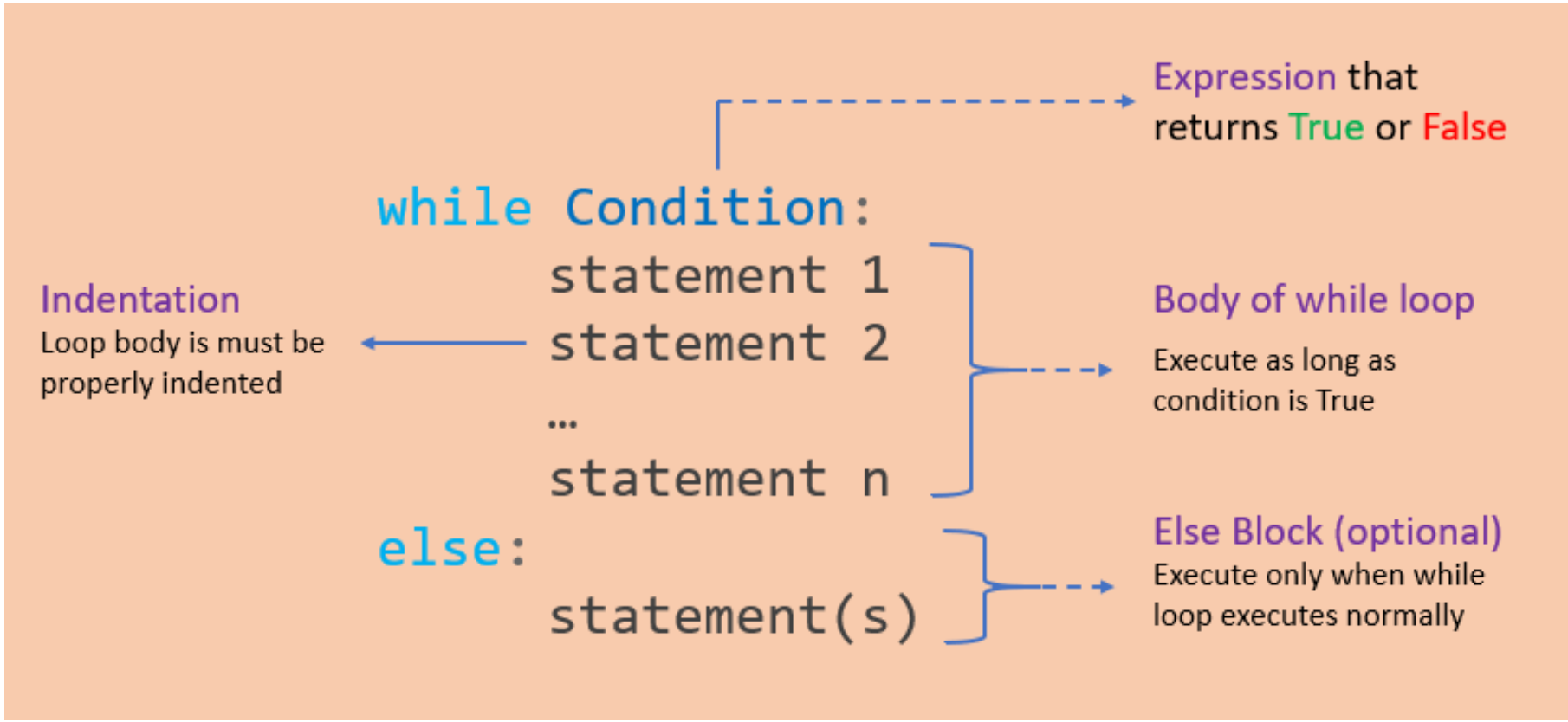
Logical operators

Operator	Description	Example
and (Logical and)	True if both the operands are True	a and b
or (Logical or)	True if either of the operands is True	a or b
not (Logical not)	True if the operand is False	not a

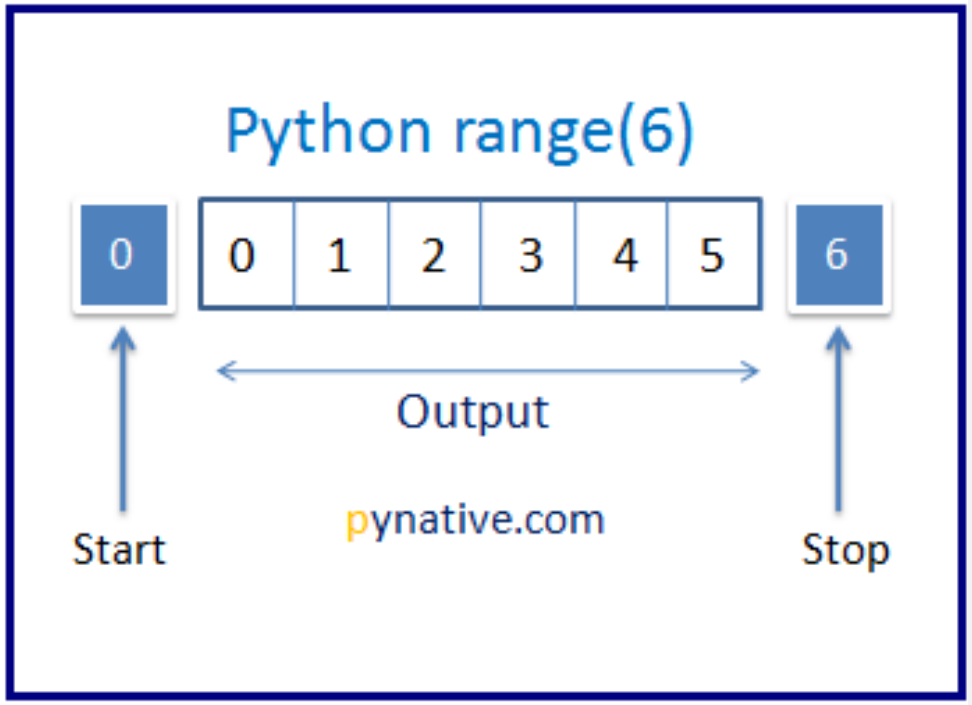
For Loops


Definite iterations.
(Total 5 iterations)
for i in range(5):
statement 1
statement 2
…
statement n
else:
statement(s)
Indentation
Loop body is must be properly indented
Body of for loop
Execute till the last item of a sequence
Else Block (optional)
Execute only when for loop executes normally

While Loops


Expression that
returns True or False
while Condition:
statement 1
statement 2
…
statement n
else:
statement(s)
Indentation
Loop body is must be
properly indented
Body of while loop
Execute as long as
condition is True
Else Block (optional)
Execute only when while
loop executes normally

Range Function



How to use Python range(start, stop, step)

range() returns the immutable sequence of numbers starting from the given **start integer to the stop-step**. Each number is incremented by adding step value to its preceding number

range(0, 6, 1) → 0 1 2 3 4 5

Step. (Optional) Specify the increment. Default is **1**

Stop. (**Required**) specifying at which position to stop. Not part of the result

Start. (Optional) Start number of sequence. Default is **0**

```
for i in range(6):
    print(i)
```

It returns a range object not list
type(range(6)) -> class 'range'

Reverse range/ Decrementing

1. range(5, -1, -1) → 5 4 3 2 1 0
2. reversed(range(6))

List from range x = **list**(range(6)) → 0 1 2 3 4 5

range(-1, -11, -1)	Negative range from -1 to -10
range(start, stop+step, step)	Generate an inclusive range
range(0, 10)[5]	Access 5th number of a range()
range(10)[3:8]	Slice a range to from index 3 to 8
range(5).start, range(5).stop	Access range() attributes

Range Function

```
>>> # One parameter
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

```
>>> # Two parameters
>>> for i in range(3, 6):
...     print(i)
...
3
4
5
```

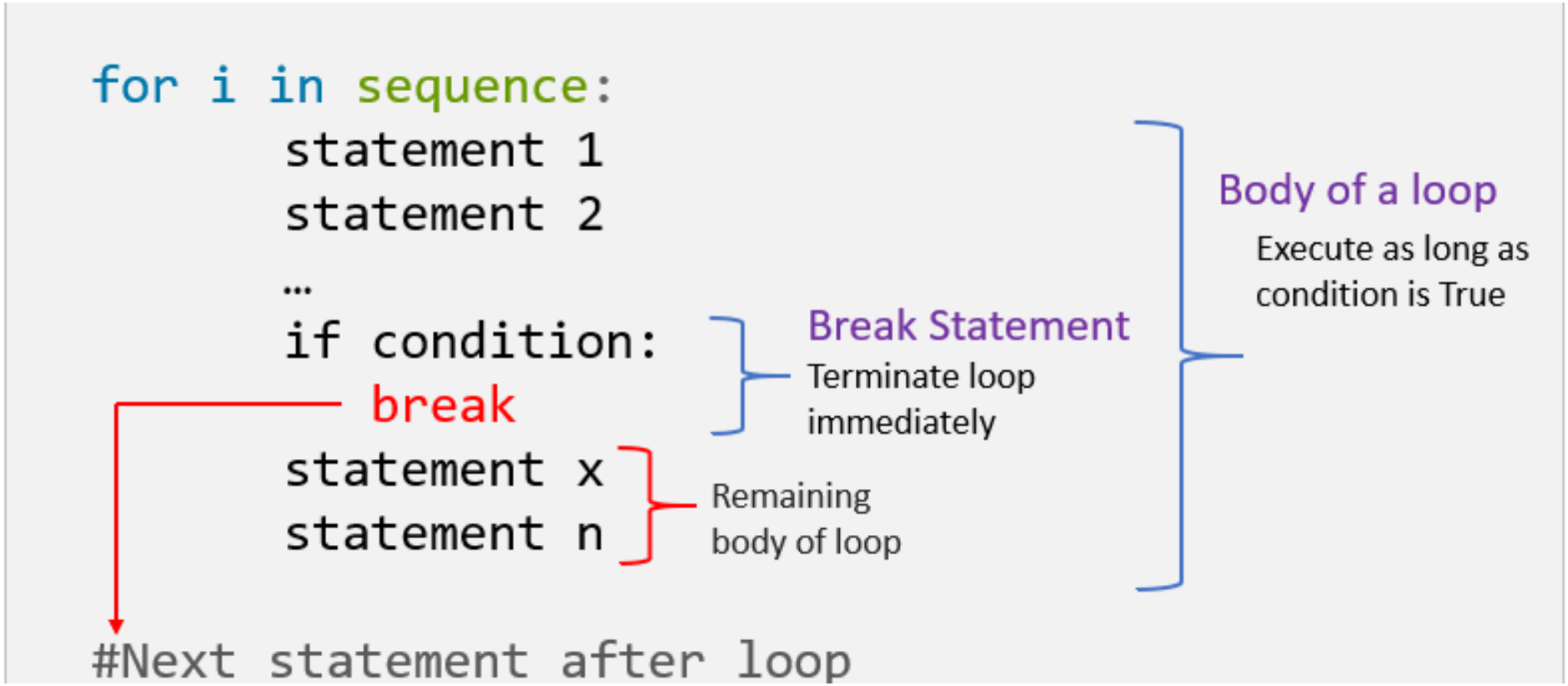
```
>>> # Three parameters
>>> for i in range(4, 10, 2)
...     print(i)
...
4
6
8
```

```
>>> # Going backwards
>>> for i in range(0, -10, -2):
...     print(i)
...
0
-2
-4
-6
-8
```

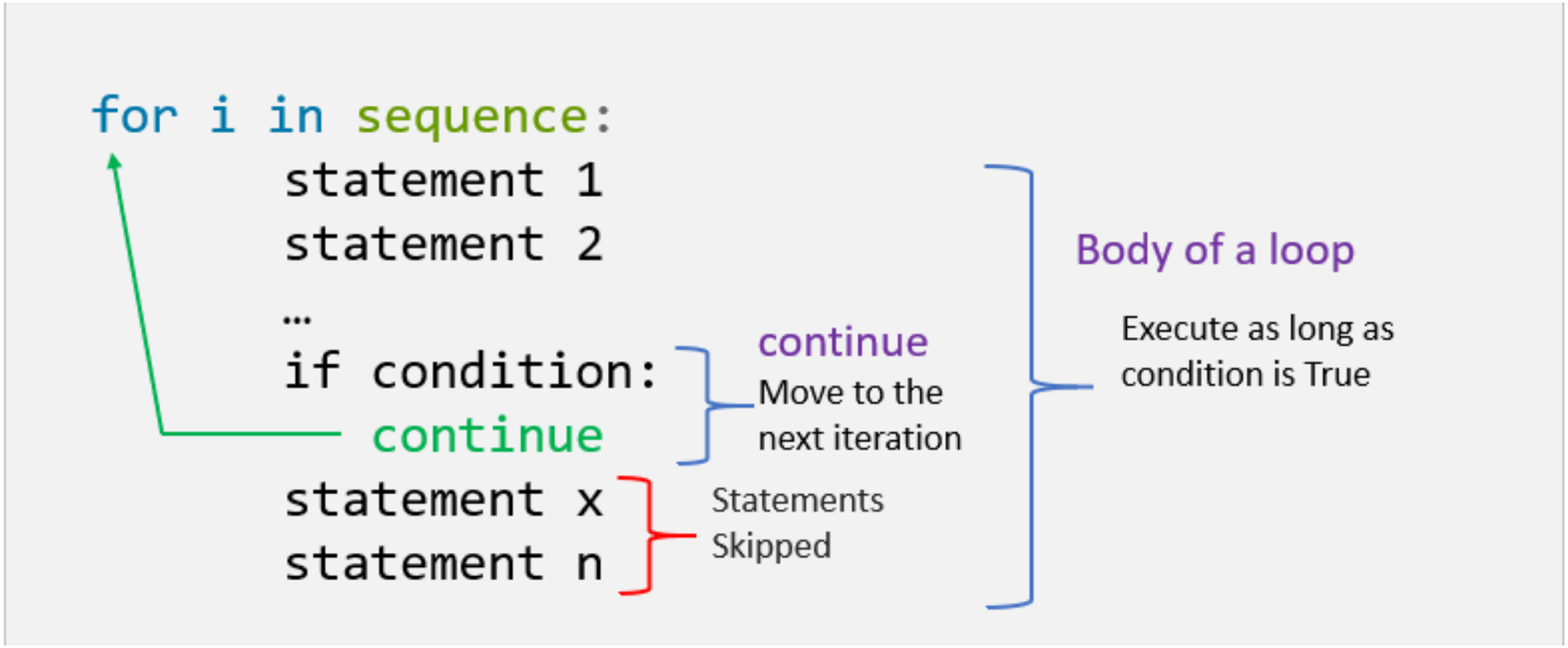
Break, Continue, and Pass

Statement	Description
break	Terminate the current loop. Use the break statement to come out of the loop instantly.
continue	Skip the current iteration of a loop and move to the next iteration
pass	Do nothing. Ignore the condition in which it occurred and proceed to run the program as usual

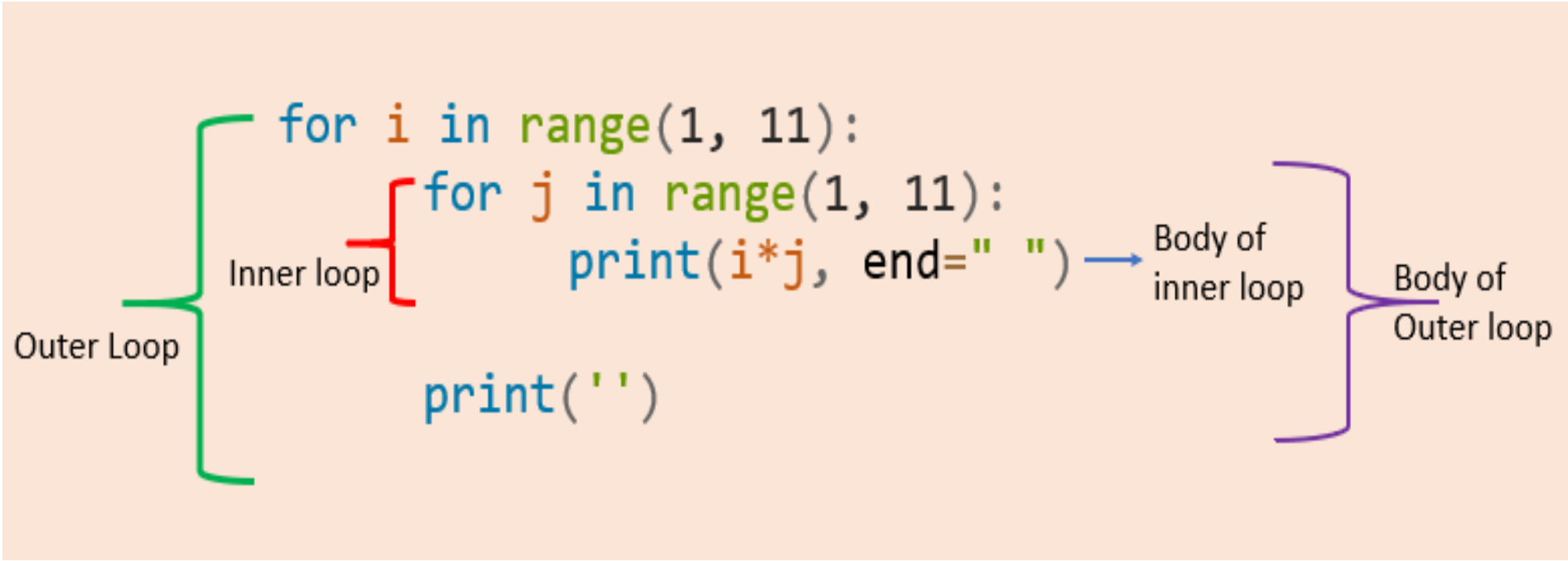
break


for i in sequence:
statement 1
statement 2
…
if condition:
break
statement x
statement n
#Next statement after loop
Break Statement
Terminate loop immediately
Remaining body of loop
Body of a loop
Execute as long as condition is True

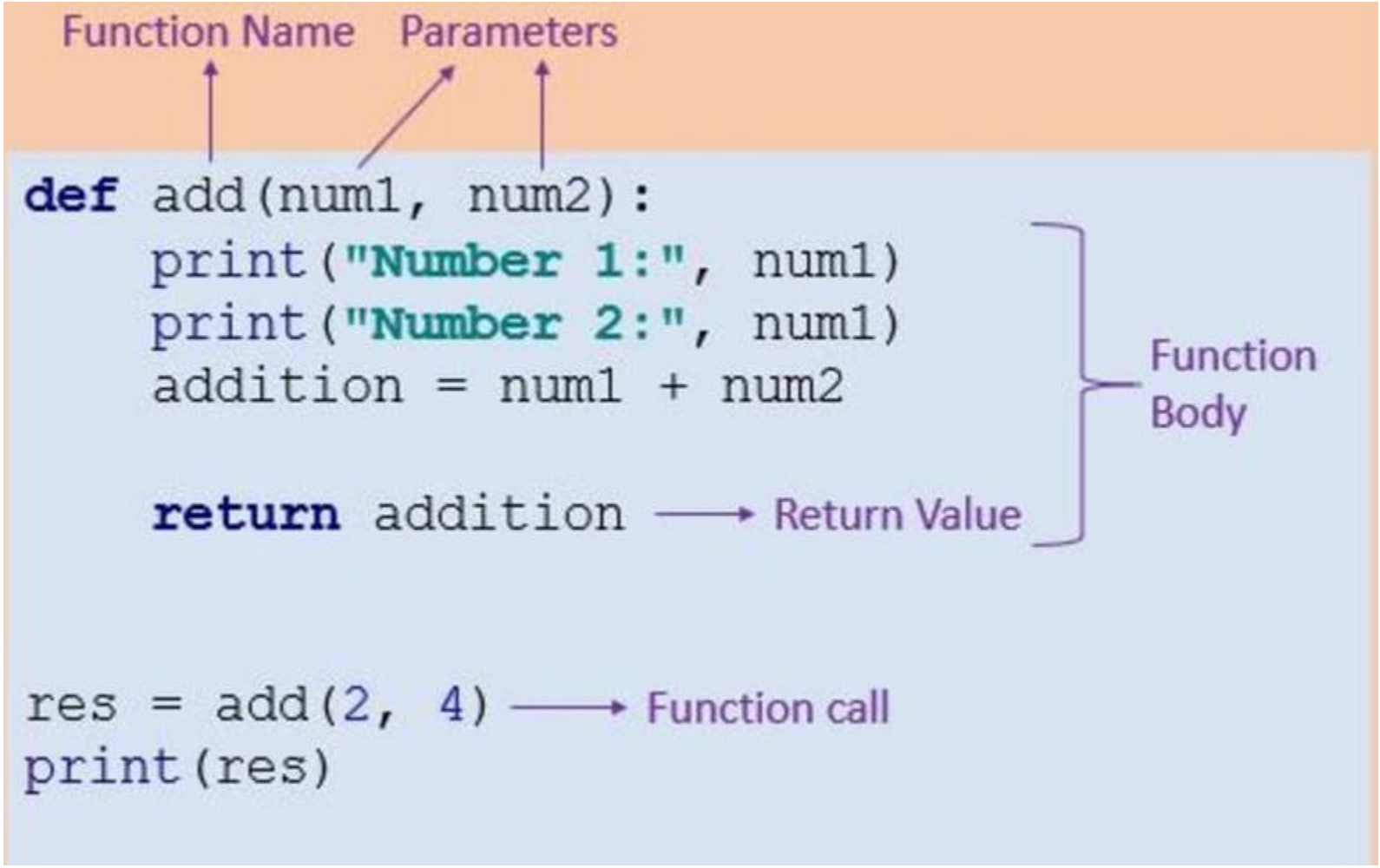
continue


for i in sequence:
statement 1
statement 2
…
if condition:
continue
statement x
statement n
continue
Move to the next iteration
Statements Skipped
Body of a loop
Execute as long as condition is True

Nested Loop


for i in range(1, 11):
for j in range(1, 11):
print(i*j, end=" ")
print('')
Outer Loop
Inner loop
Body of inner loop
Body of Outer loop

Functions


Function Name
Parameters
def add(num1, num2):
print("Number 1:", num1)
print("Number 2:", num1)
addition = num1 + num2
return addition
Return Value
Function Body
res = add(2, 4)
Function call
print(res)

Reading and Writing to text files in Python

File Access Modes

1. **Read Only (‘r’) :** Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened.
2. **Read and Write (‘r+’) :** Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists.
3. **Write Only (‘w’) :** Open the file for writing. For existing file, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exists.
4. **Write and Read (‘w+’)** : Open the file for reading and writing. For existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.
5. **Append Only (‘a’)** : Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
6. **Append and Read (‘a+’) :** Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

Opening a File

```
File_object = open(r"File_Name","Access_Mode")
```

```
# Open function to open the file "MyFile1.txt"
# (same directory) in append mode and
file1 = open("MyFile.txt","a")

# store its reference in the variable file1
# and "MyFile2.txt" in D:\Text in file2
file2 = open(r"D:\Text\MyFile2.txt","w+")
```

Closing a file

```
# Opening and Closing a file "MyFile.txt"
# for object name file1.
file1 = open("MyFile.txt","a")
file1.close()
```

Writing to a file

There are two ways to write in a file.

1. write() : Inserts the string str1 in a single line in the text file.

File_object.write(str1)

2. writelines() : For a list of string elements, each string is inserted in the text file. Used to insert multiple strings at a single time.

File_object.writelines(L) for L = [str1, str2, str3]

Reading from a file

There are three ways to read data from a text file.

1.read() : Returns the read bytes in form of a string. Reads n bytes, if no n specified, reads the **entire file**.

File_object.read([n])

2.readline() : Reads a line of the file and returns in form of a string. For specified n, reads at most n bytes.

However, does not reads more than **one line**, even if n exceeds the length of the line.

File_object.readline([n])

3.readlines() : Reads **all the lines** and return them as each line a string element in a **list**.

File_object.readlines()

Using write along with the with() function

Example

```
# Python code to illustrate split() function
with open("file.txt", "r") as file:
        data = file.readlines()
        for line in data:
                word = line.split()
                print (word)
```

Copy contents of one file to another file

```
# open both files
with open('first.txt','r') as firstfile, open('second.txt','a') as secondfile:

    # read content from first file
    for line in firstfile:

        # append content to second file
        secondfile.write(line)
```

Copy all the content of one file to another file in uppercase

```
# To open the first file in read mode
f1 = open("sample file 1.txt", "r")

# To open the second file in write mode
f2 = open("sample file 2.txt", "w")

# For loop to traverse through the file
for line in f1:

        # Writing the content of the first
        # file to the second file

        # Using upper() function to
        # capitalize the letters
        f2.write(line.upper())
```

Copy odd lines of one file to other

```
# open file in read mode
fn = open('bcd.txt', 'r')
# open other file in write mode
fn1 = open('nfile.txt', 'w')
# read the content of the file line by line
cont = fn.readlines()
type(cont)
for i in range(0, len(cont)):
        if(i % 2 ! = 0):
                fn1.write(cont[i])
        else:
                pass
```

Count number of lines in a text file in Python

```
file = open("gfg.txt","r")
Counter = 0

# Reading from file
Content = file.read()
CoList = Content.split("\n")

for i in CoList:
        if i:
                Counter += 1
print("This is the number of lines in the file")
print(Counter)
```

Assignment

- Get number of characters, words, spaces and lines in a file
- reverse the content of a file and store it in another file

Modules

1. Built-in Modules
2. User-defined Modules

Modules

- Datetime
- Math
- Random
- Pandas
- numpy

THANK YOU