Machine Learning Diploma

Session 2: Setup Environment & Python Basics

Agenda:

1	Data types (One variable : Many Data)
2	python operations
3	If condition
4	loops

1- Data types (One variable : Many Data)

Python Data Types

One variable : One Data

Integer	Float	String	Boolean
whole number without a decimal point.	numeric data type that represents real numbers and can include a decimal point.	sequence of characters, enclosed within single or double quotes, used to represent text data.	binary data type representing either True or False

One variable : Many Data

List	Tuple	Set	Dictionary
1. Data ordered 2. Changeable Data 3. Allow Duplicate Data 4. List can be represented by [ ] 5. Can be nested among all 6. Convert any datatype to list using List() function	1. Data ordered 2. Unchangeable Data 3. Allow Duplicate Data 4. Tuple can be represented by ( ) 5. Can be nested among all 6. Convert any datatype to tuple using tuple() function	1. Data unordered 2. Unchangeable Data 3. Not allow Duplicate Data 4. Tuple can be represented by { } 5. Can be nested among all 6. Convert any datatype to set using set() function	1. Data ordered 2. Changeable Data 3. Not allow Duplicate for keys 4. Dictionary can be represented by { } 5. Can be nested among all 6. Convert any datatype to dictionary using dict() function

List

List

1. **List can be represented by [ ]**

```
list_one = [1,2,3,4,"amit","learning",True,[1,2,3,"True",["ml","dl","ds"]]]

print(type(list_one))
```

```
<class 'list'>
```

2. Ordered

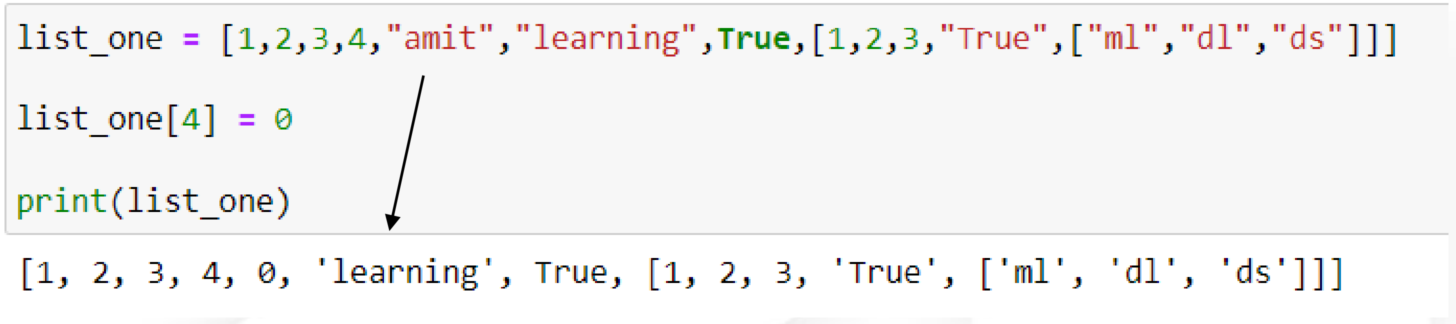
(element in the list is assigned a specific index or position.)

```
list_one = [1,2,3,4,"amit","learning",True,[1,2,3,"True",["ml","dl","ds"]]]

print(list_one[4])
```

```
amit
```

List

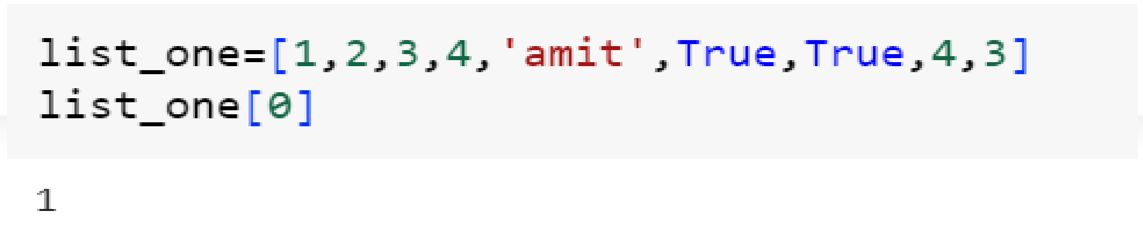
3. Changeable Data



```
list_one = [1,2,3,4,"amit","learning",True,[1,2,3,"True",["ml","dl","ds"]]]

list_one[4] = 0

print(list_one)
```

```
[1, 2, 3, 4, 0, 'learning', True, [1, 2, 3, 'True', ['ml', 'dl', 'ds']]]
```

4. Allow Duplicate Data



```
list_one=[1,2,3,4,'amit',True,True,4,3]
list_one[0]
```

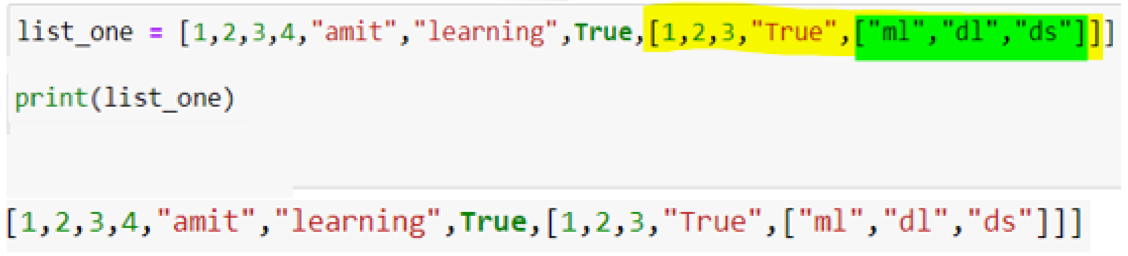
```
1
```

AMIT

List

5. Can be nested among all

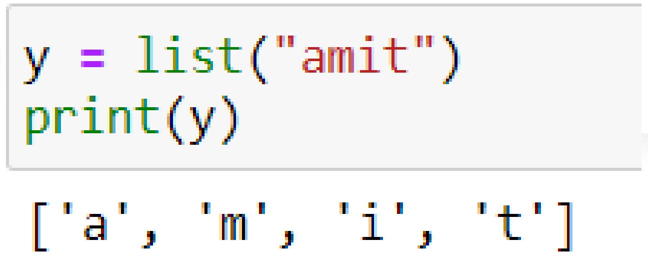
(you can have lists within lists.)



```
list_one = [1,2,3,4,"amit","learning",True,[1,2,3,"True",["ml","dl","ds"]]]

print(list_one)
```

```
[1,2,3,4,"amit","learning",True,[1,2,3,"True",["ml","dl","ds"]]]
```

6. Convert any datatype to list using List() function



```
y = list("amit")
print(y)
```

```
['a', 'm', 'i', 't']
```

Add Item To List

append()

list.append(item)

adds a specified element to the end of the list.

insert()

list.insert(index)

adds a specified element at a given index, shifting existing elements.

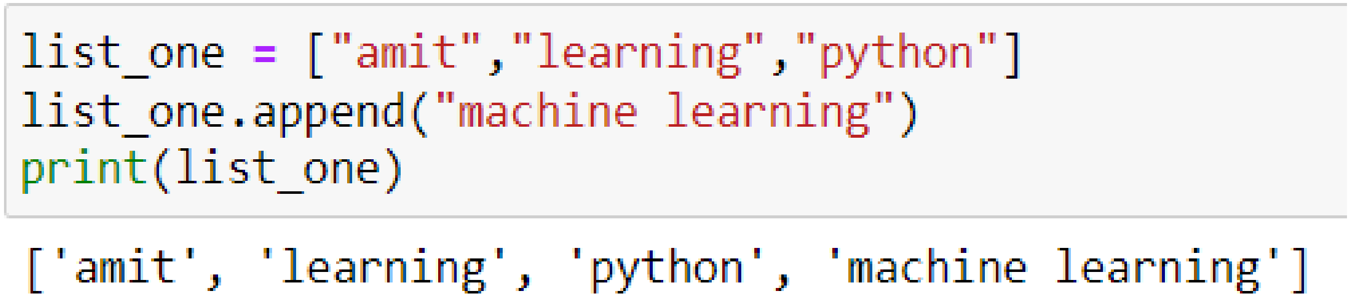
extend()

list.extend(index)

appends elements from an iterable to the end of an existing list.

Add Item To List

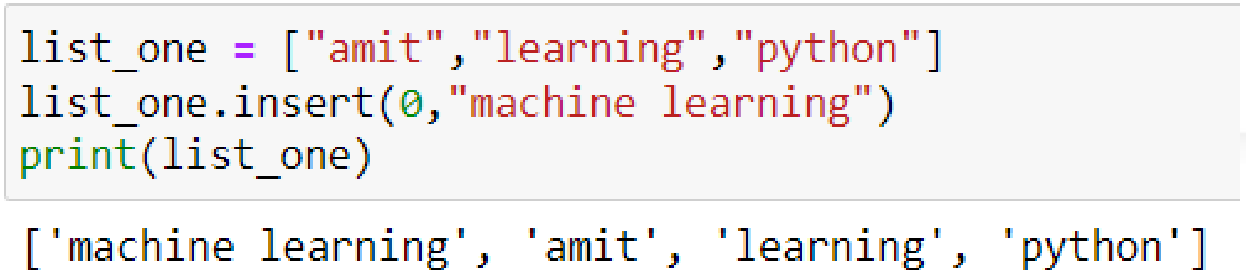
1. Append(): Add the specified item to the end of the list.



```
list_one = ["amit","learning","python"]
list_one.append("machine learning")
print(list_one)
```

```
['amit', 'learning', 'python', 'machine learning']
```

2. insert(): adds the specified item at the given index in the list.

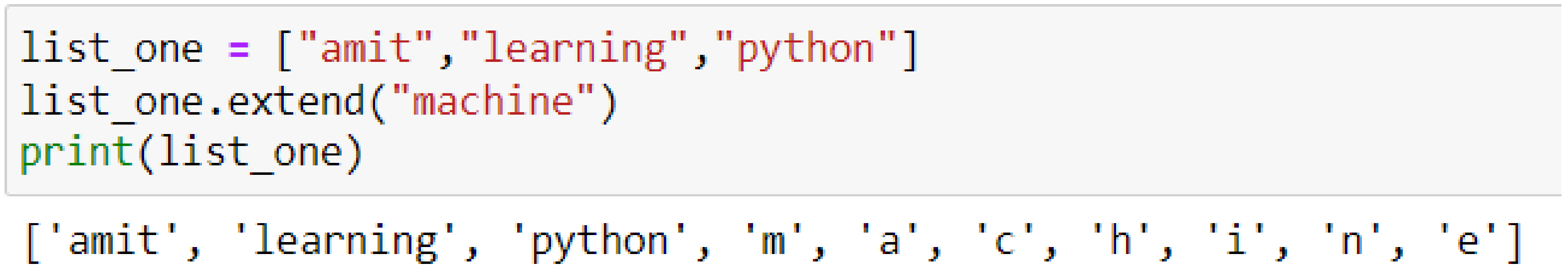


```
list_one = ["amit","learning","python"]
list_one.insert(0,"machine learning")
print(list_one)
```

```
['machine learning', 'amit', 'learning', 'python']
```

Add Item To List

3. extend(): adds elements from the iterable to the end of the list.



```python
list_one = ["amit","learning","python"]
list_one.extend("machine")
print(list_one)
```

```
['amit', 'learning', 'python', 'm', 'a', 'c', 'h', 'i', 'n', 'e']
```

delete Item To List

remove()

list.remove(item)

removes the first occurrence of the specified item from the list.

pop()

list.pop(index)

removes and returns the item at the specified index in the list.

del()

del list[index]

removes the item at the specified index using the del statement.

Delete Item To List

1. **remove(): removes the first occurrence of the specified item from the list.**

```
list_one = ["amit","learning","python"]
list_one.remove("learning")
print(list_one)
```

```
['amit', 'python']
```

2. **pop(): removes and returns the item at the specified index in the list.**

```
list_one = ["amit","learning","python"]
list_one.pop()
print(list_one)
```

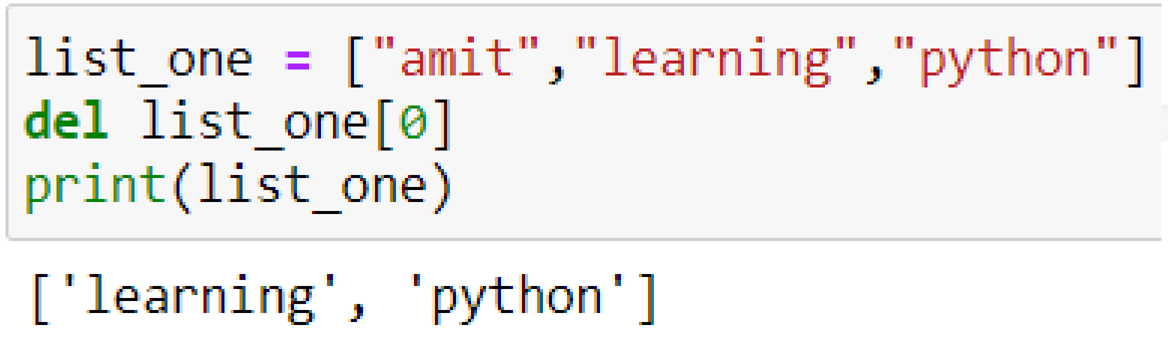
```
['amit', 'learning']
```

```
list_one = ["amit","learning","python"]
list_one.pop(0)
print(list_one)
```

```
['learning', 'python']
```

Delete Item To List

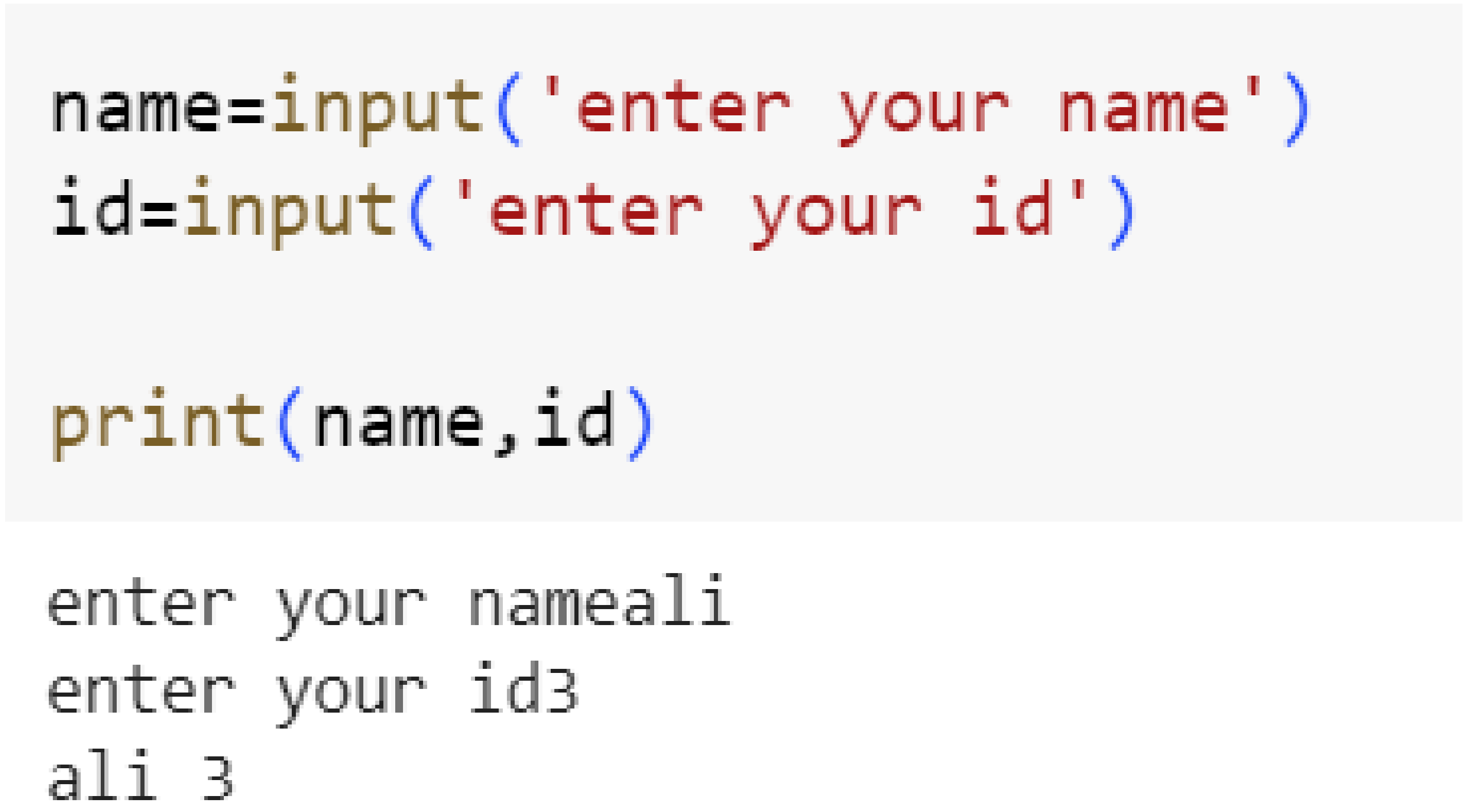
3. del : removes the item at the specified index using the del statement.



```
list_one = ["amit","learning","python"]
del list_one[0]
print(list_one)
```

```
['learning', 'python']
```

Input function



```python
name=input('enter your name')
id=input('enter your id')

print(name,id)
```

```
enter your nameali
enter your id3
ali 3
```

Try To Solve

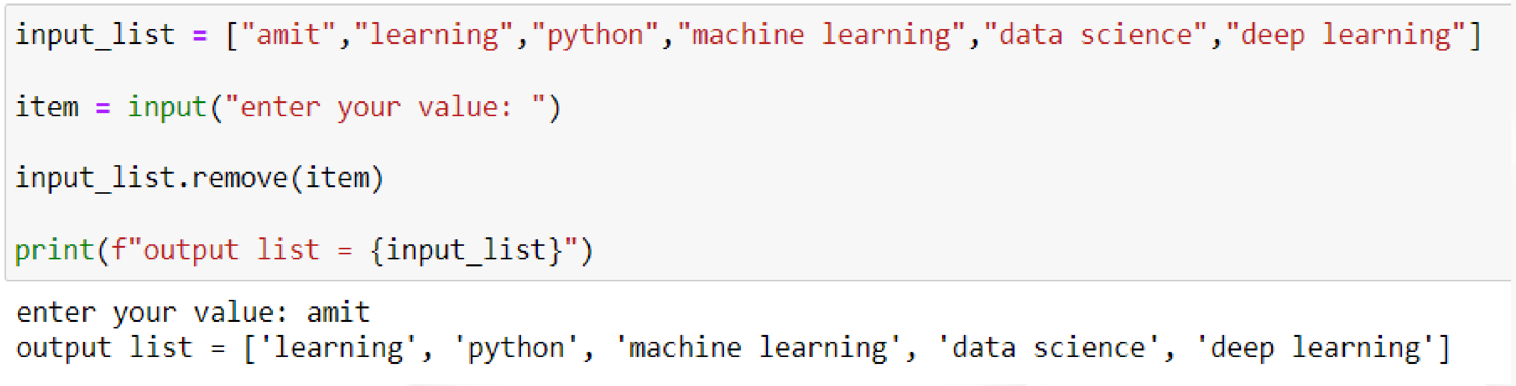
Write Python code snippet removes a user-inputted item from a list and outputs the modified list?

Input_list: ["amit","learning","python","machine learning","data science","deep learning"]

Try To Solve

Write Python code snippet removes a user-inputted item from a list and outputs the modified list?

Input_list: ["amit","learning","python","machine learning","data science","deep learning"]

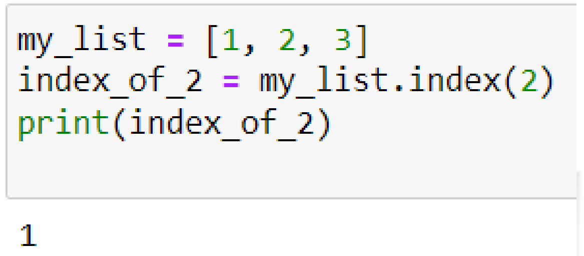


```python
input_list = ["amit","learning","python","machine learning","data science","deep learning"]

item = input("enter your value: ")

input_list.remove(item)

print(f"output list = {input_list}")
```

```
enter your value: amit
output list = ['learning', 'python', 'machine learning', 'data science', 'deep learning']
```

Functions related to list

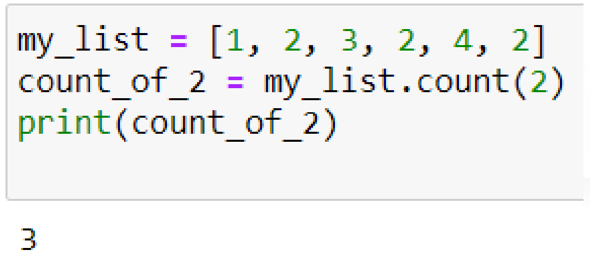
index() : Returns the index of the first occurrence of a specified value.



```python
my_list = [1, 2, 3]
index_of_2 = my_list.index(2)
print(index_of_2)
```

```
1
```

count() : Returns the number of occurrences of a specified value.

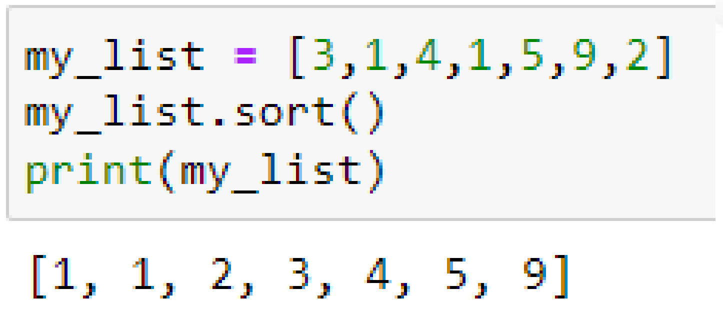


```python
my_list = [1, 2, 3, 2, 4, 2]
count_of_2 = my_list.count(2)
print(count_of_2)
```

```
3
```

Functions related to list

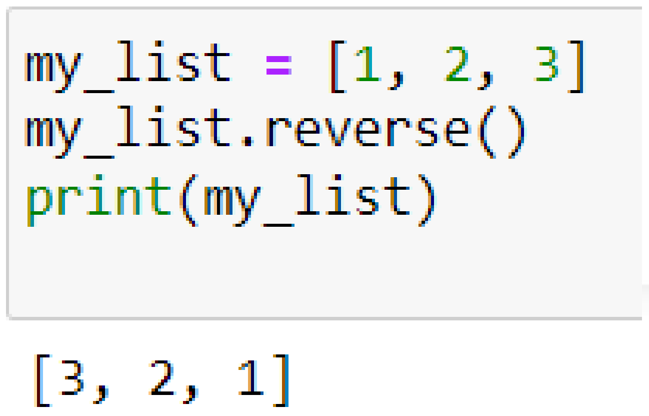
sort() : Sorts the list in ascending order.



```
my_list = [3,1,4,1,5,9,2]
my_list.sort()
print(my_list)
```

```
[1, 1, 2, 3, 4, 5, 9]
```

reverse() : Reverses the order of the list.

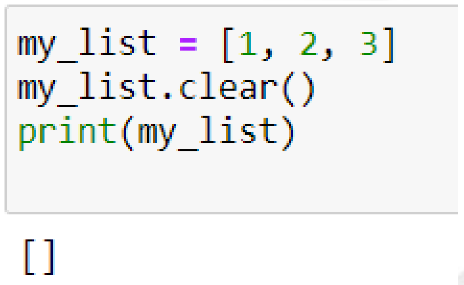


```
my_list = [1, 2, 3]
my_list.reverse()
print(my_list)
```

```
[3, 2, 1]
```

Functions related to list

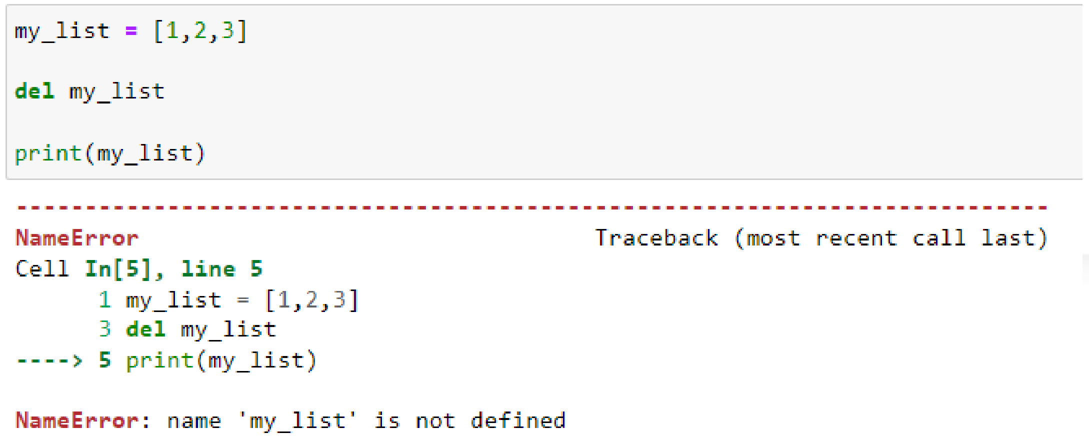
clear() : Removes all elements from the list.



```
my_list = [1, 2, 3]
my_list.clear()
print(my_list)
```

```
[]
```

del : delete the whole list from memory.



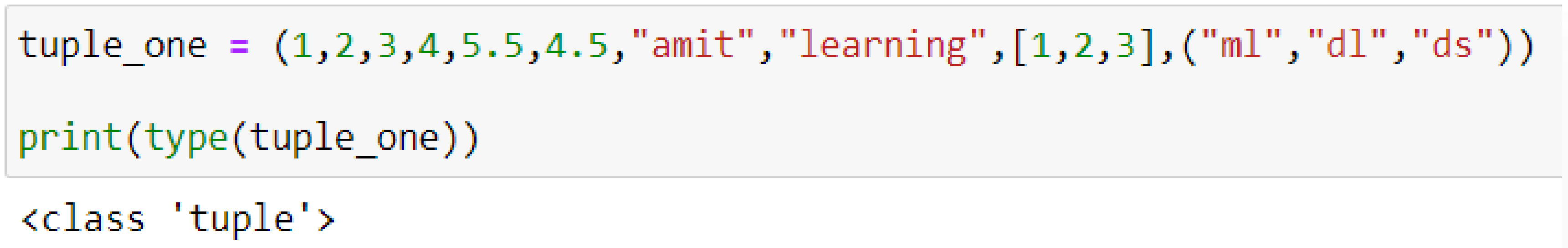
```
my_list = [1,2,3]

del my_list

print(my_list)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 5
      1 my_list = [1,2,3]
      3 del my_list
----> 5 print(my_list)

NameError: name 'my_list' is not defined
```

Tuple

Tuple

1. **Tuple can be represented by ( )**

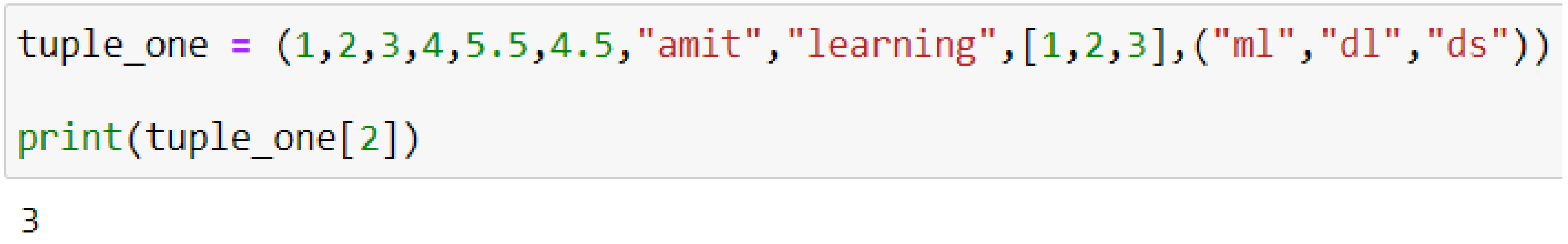


```
tuple_one = (1,2,3,4,5.5,4.5,"amit","learning",[1,2,3],("ml","dl","ds"))

print(type(tuple_one))
```

```
<class 'tuple'>
```

2. Ordered

(element in the list is assigned a specific index or position.)



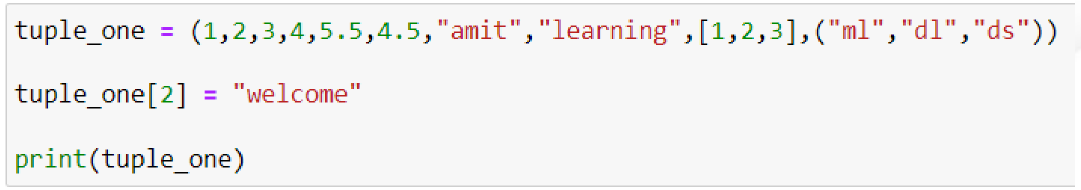
```
tuple_one = (1,2,3,4,5.5,4.5,"amit","learning",[1,2,3],("ml","dl","ds"))

print(tuple_one[2])
```

```
3
```

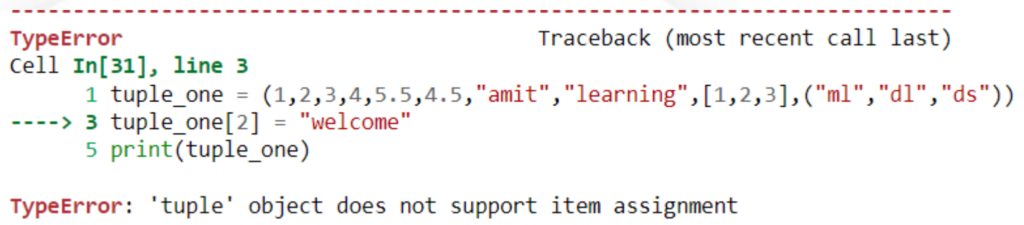
AMIT

Tuple

3. Unchangeable Data

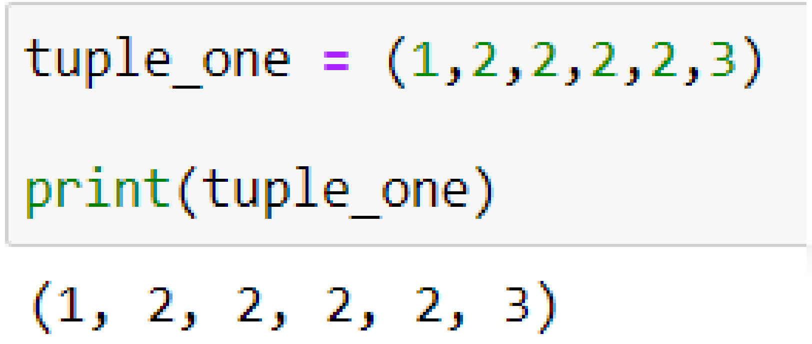


```
tuple_one = (1,2,3,4,5.5,4.5,"amit","learning",[1,2,3],("ml","dl","ds"))

tuple_one[2] = "welcome"

print(tuple_one)
```



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[31], line 3
      1 tuple_one = (1,2,3,4,5.5,4.5,"amit","learning",[1,2,3],("ml","dl","ds"))
----> 3 tuple_one[2] = "welcome"
      5 print(tuple_one)

TypeError: 'tuple' object does not support item assignment
```

4. Allow Duplicate Data

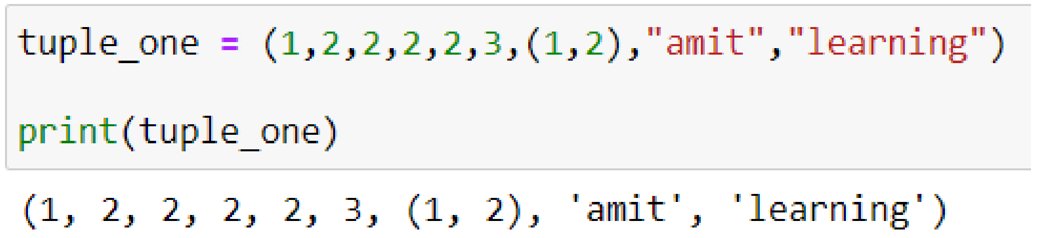


```
tuple_one = (1,2,2,2,2,3)

print(tuple_one)
```

```
(1, 2, 2, 2, 2, 3)
```

Tuple

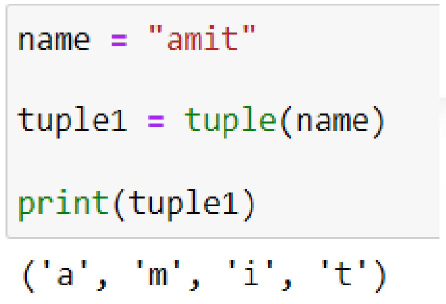
5. Can be nested among all (you can have lists within lists.)



```
tuple_one = (1,2,2,2,2,3,(1,2),"amit","learning")

print(tuple_one)
```

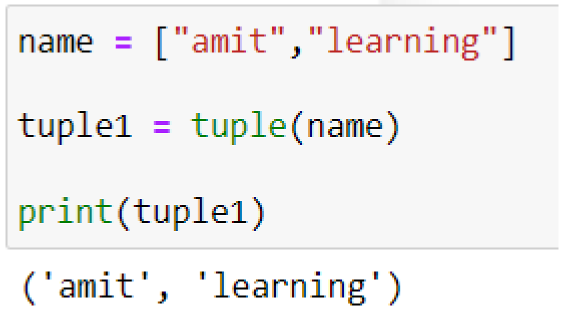
```
(1, 2, 2, 2, 2, 3, (1, 2), 'amit', 'learning')
```

6. Convert any datatype to list using Tuple() function



```
name = "amit"

tuple1 = tuple(name)

print(tuple1)
```

```
('a', 'm', 'i', 't')
```



```
name = ["amit","learning"]

tuple1 = tuple(name)

print(tuple1)
```

```
('amit', 'learning')
```

Add Item To Tuple

Convert tuple to list (Tuple Unchangeable)

Add Item To List

Append()

list.append(item)

adds the specified item to the end of the list.

insert()

list.insert(index, item)

adds the specified item at the given index in the list.

extend()

list.extend(item)

adds elements from the iterable to the end of the list.

Add Item To Tuple

Convert tuple to list (Tuple Unchangeable)

Add Item To List

pop()

list.pop(index)

removes and returns the item at the specified index in the list.

remove()

list.remove(item)

removes the first occurrence of the specified item from the list.

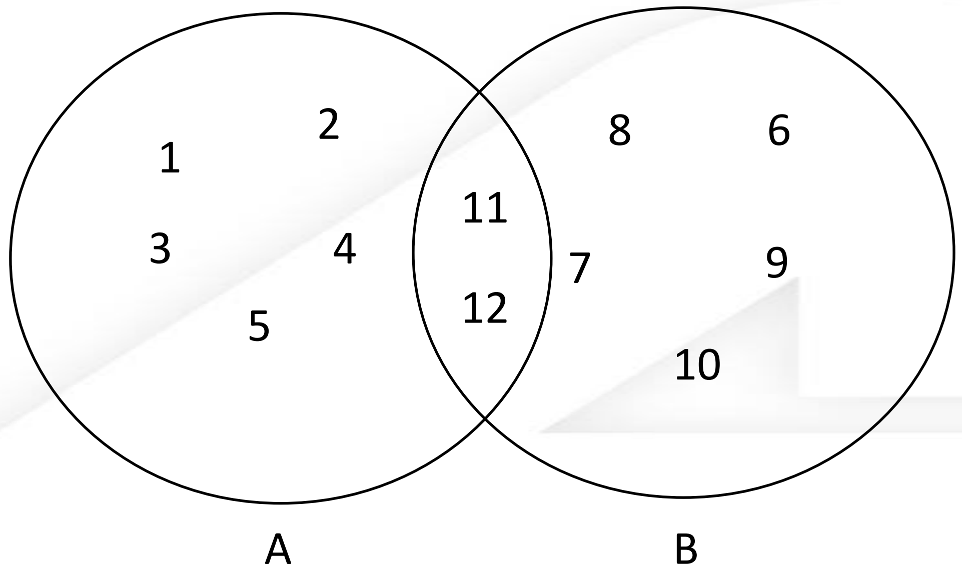
del()

del list[index]

removes the item at the specified index using the del statement.

Set

Set
Recap



Set A = {1,2,3,4,5,11,12} or {1,2,3,4, 12,5,11} or {1,3,4, 12,5,11,2} → (data not necessary to be ordered)

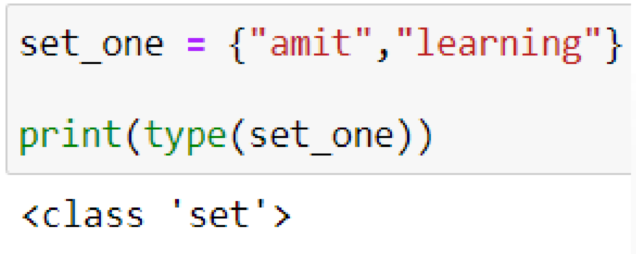
Set B = {8,6,7,9,10,11,12} or {8,11,12,6,7,9,10} or {8,11,9,10,12,6,7} → (data not necessary to be ordered)

Set A ∩ Set B = {11,12}

Set A ∪ Set B = {11,12}

Set

1. **Set can be represented by { }**

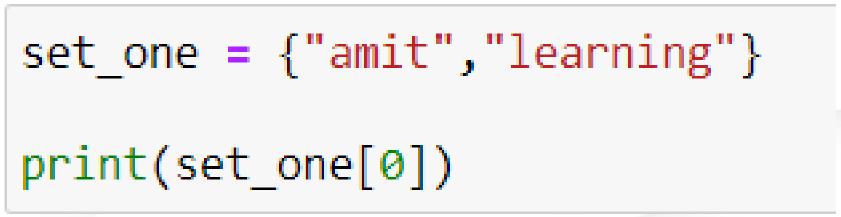


```
set_one = {"amit","learning"}

print(type(set_one))
```

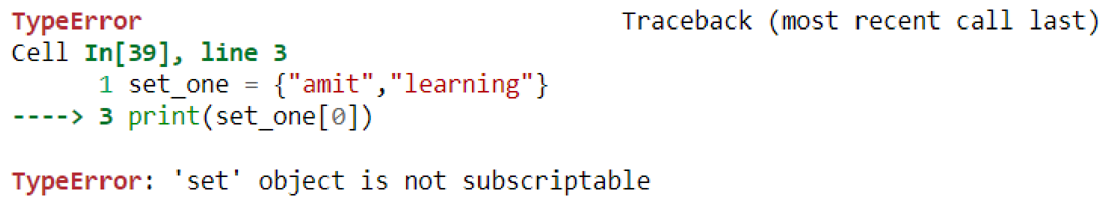
```
<class 'set'>
```

2. Unordered

(element in the list is not assigned a specific index or position.)



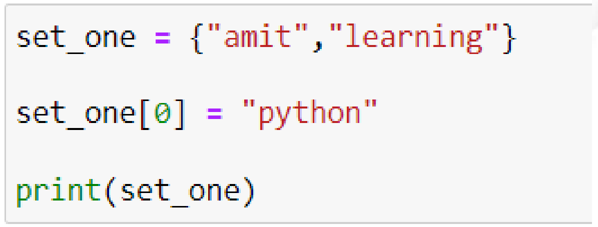
```
set_one = {"amit","learning"}

print(set_one[0])
```



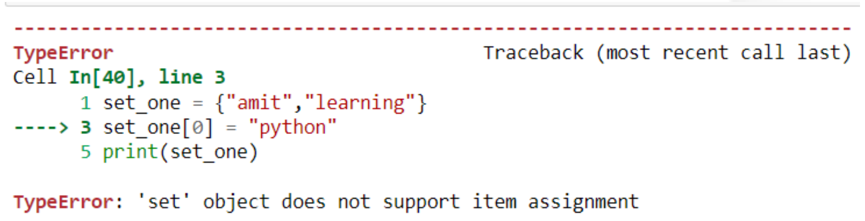
```
TypeError                                 Traceback (most recent call last)
Cell In[39], line 3
      1 set_one = {"amit","learning"}
----> 3 print(set_one[0])

TypeError: 'set' object is not subscriptable
```

set

3. Unchangeable Data

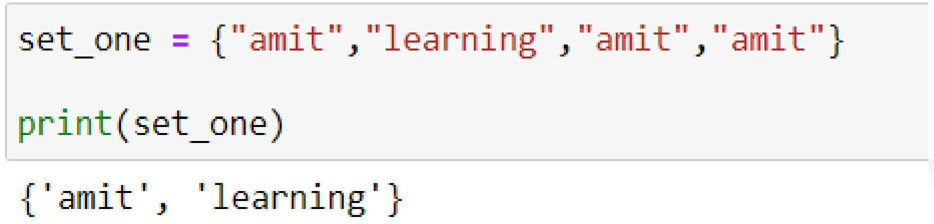


```
set_one = {"amit","learning"}

set_one[0] = "python"

print(set_one)
```



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 3
      1 set_one = {"amit","learning"}
----> 3 set_one[0] = "python"
      5 print(set_one)

TypeError: 'set' object does not support item assignment
```

4. Not allow Duplicate Data

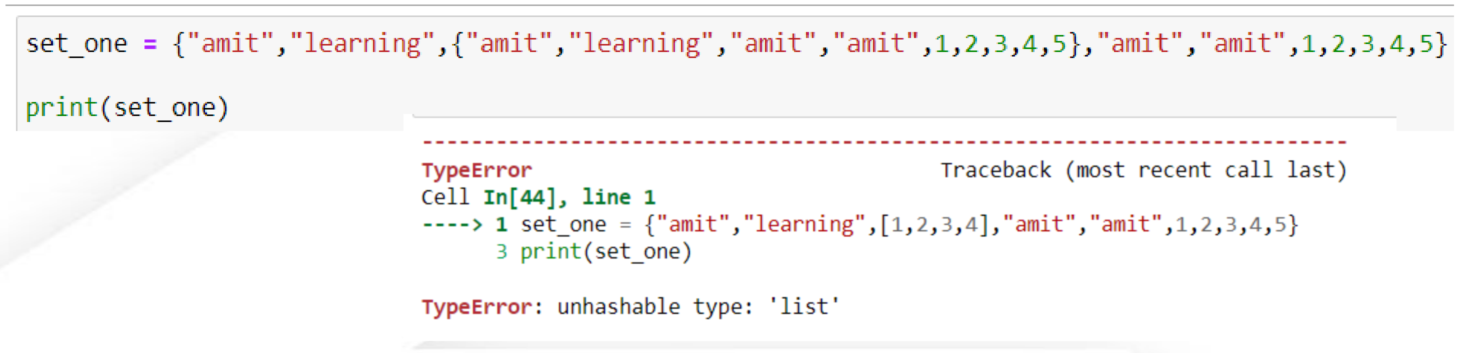


```
set_one = {"amit","learning","amit","amit"}

print(set_one)
```

```
{'amit', 'learning'}
```

set

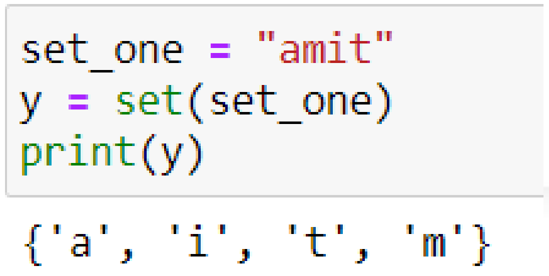
5. Can not be nested among all
(you can not have lists within lists.)



```
set_one = {"amit","learning",{"amit","learning","amit","amit",1,2,3,4,5},"amit","amit",1,2,3,4,5}

print(set_one)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 1
----> 1 set_one = {"amit","learning",[1,2,3,4],"amit","amit",1,2,3,4,5}
      3 print(set_one)

TypeError: unhashable type: 'list'
```

6. Convert any datatype to list using set() function



```
set_one = "amit"
y = set(set_one)
print(y)
```

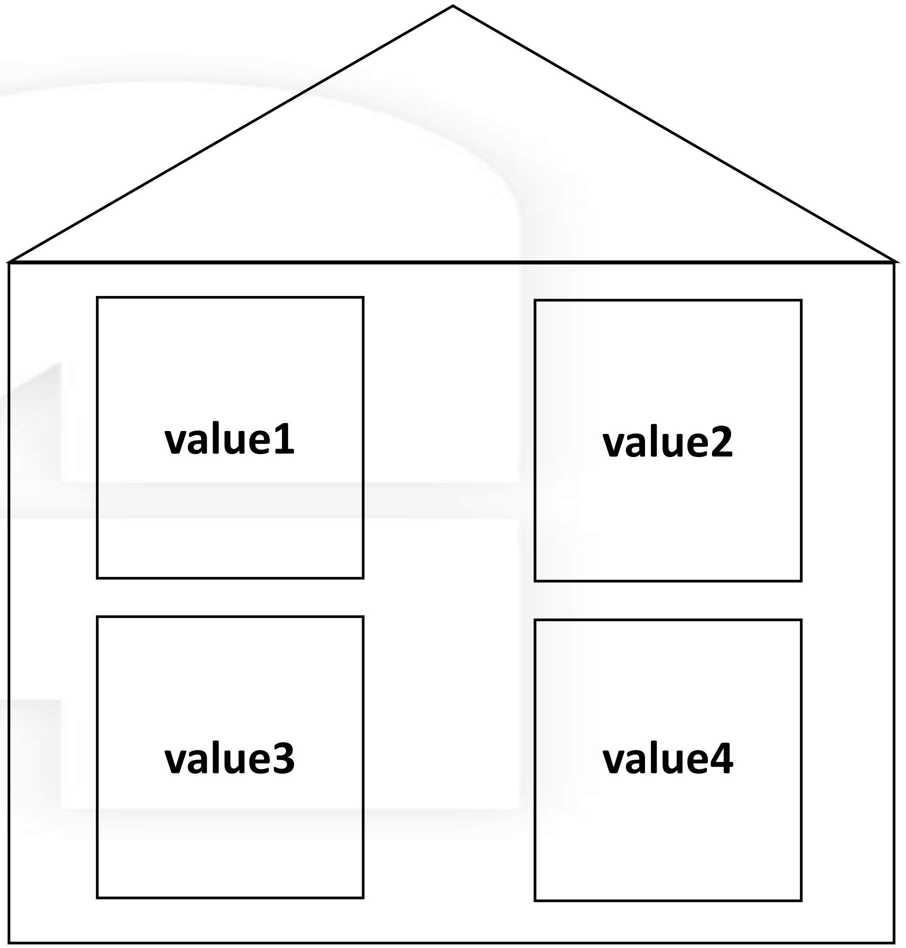
```
{'a', 'i', 't', 'm'}
```

Dictionary

Dictionary

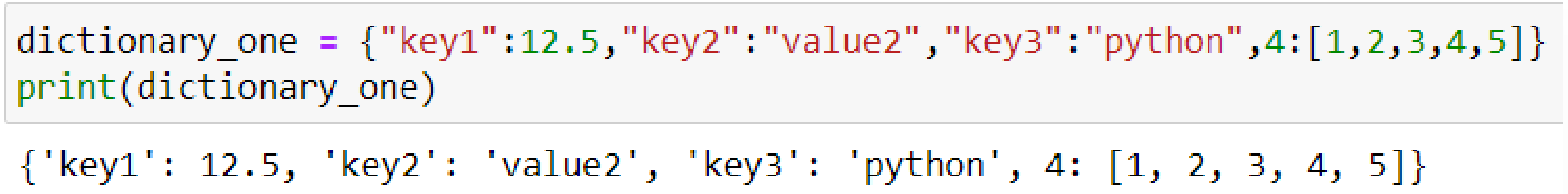
A dictionary can be likened to a structure resembling a building, where within this construct, there exist distinct houses (4 houses). Each house encapsulates data, and to access the information contained within each house, one must utilize a specific key.

Key1 ➔ access ➔ value1
Key2 ➔ access ➔ value2
Key3 ➔ access ➔ value3
Key4 ➔ access ➔ value4



Dictionary

1. **dictionary can be represented by { }**

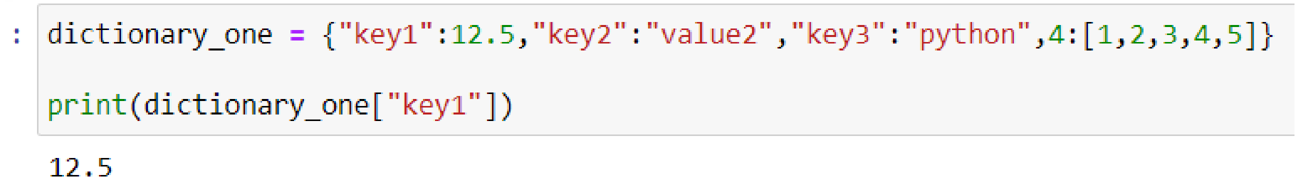


```
dictionary_one = {"key1":12.5,"key2":"value2","key3":"python",4:[1,2,3,4,5]}
print(dictionary_one)
```

```
{'key1': 12.5, 'key2': 'value2', 'key3': 'python', 4: [1, 2, 3, 4, 5]}
```

2. ordered

(element in the list is not assigned a specific index or position.)



```
: dictionary_one = {"key1":12.5,"key2":"value2","key3":"python",4:[1,2,3,4,5]}

  print(dictionary_one["key1"])
```

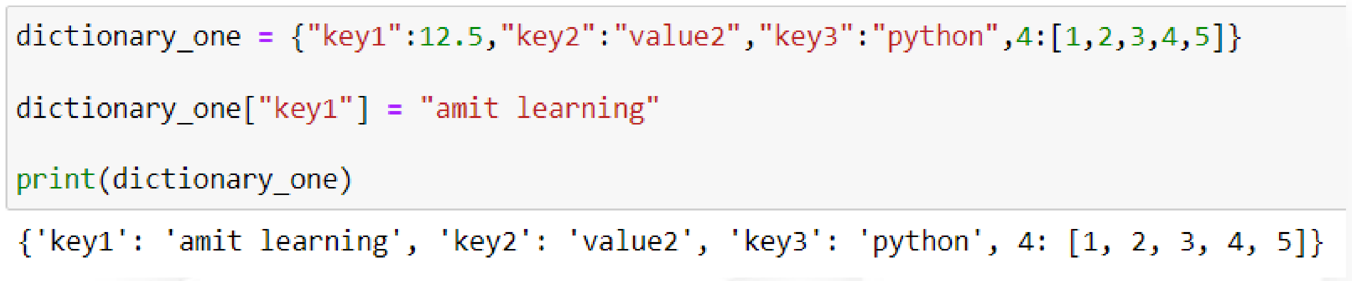
```
12.5
```

Note:

1. Dictionary keys can encompass any data type.
2. Note: Dictionary values are versatile and can accommodate any data type

Dictionary

3. changeable Data

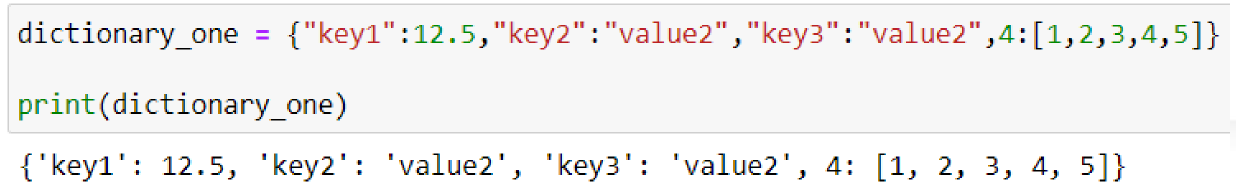


```
dictionary_one = {"key1":12.5,"key2":"value2","key3":"python",4:[1,2,3,4,5]}

dictionary_one["key1"] = "amit learning"

print(dictionary_one)
```

```
{'key1': 'amit learning', 'key2': 'value2', 'key3': 'python', 4: [1, 2, 3, 4, 5]}
```

4. allow Duplicate Data as value not as a key

(element in the list is not assigned a specific index or position.)



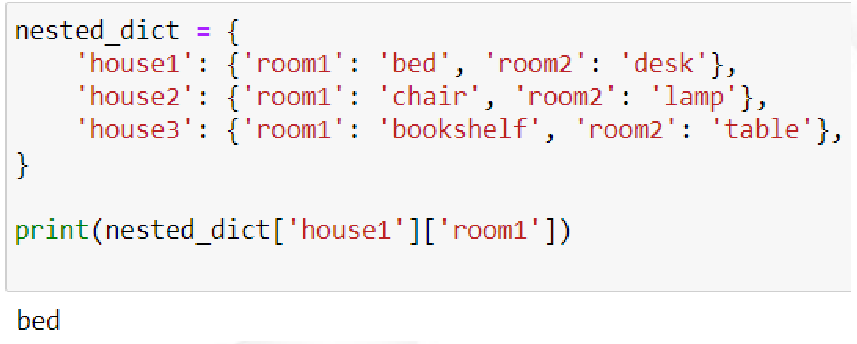
```
dictionary_one = {"key1":12.5,"key2":"value2","key3":"value2",4:[1,2,3,4,5]}

print(dictionary_one)
```

```
{'key1': 12.5, 'key2': 'value2', 'key3': 'value2', 4: [1, 2, 3, 4, 5]}
```

Dictionary

5. Can be nested among all

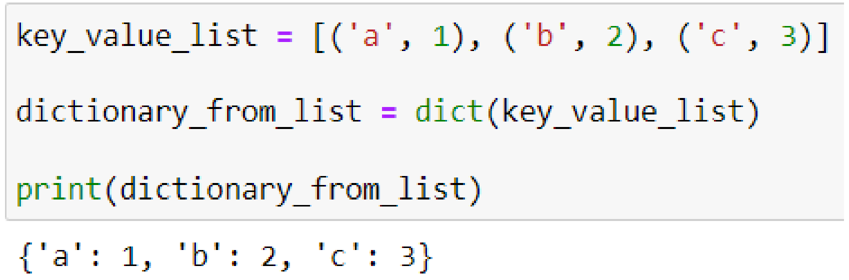
(you can have lists within lists.)



```
nested_dict = {
    'house1': {'room1': 'bed', 'room2': 'desk'},
    'house2': {'room1': 'chair', 'room2': 'lamp'},
    'house3': {'room1': 'bookshelf', 'room2': 'table'},
}

print(nested_dict['house1']['room1'])
```

```
bed
```

6. Convert any datatype to list using set() function



```
key_value_list = [('a', 1), ('b', 2), ('c', 3)]

dictionary_from_list = dict(key_value_list)

print(dictionary_from_list)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

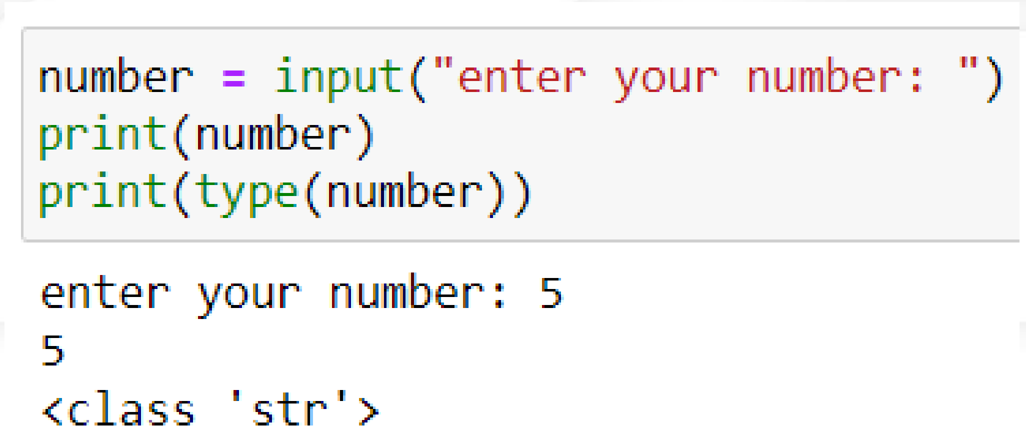
Input Function

- The input function in Python is a way to get information from the user.

- It displays a message, waits for the user to type something, and then captures and returns what the user typed.

- It's like asking a question and getting an answer from the person using your program.

- The data returned from the input function is always of string datatype.



```python
number = input("enter your number: ")
print(number)
print(type(number))
```

```
enter your number: 5
5
<class 'str'>
```

2- Python Operation

Arithmetic Operations		
Operator	Purpose	Example
+	Addition (Sum of two operands)	a + b
-	Subtraction (Difference between two operands)	a - b
*	Multiplication (Product of two operands)	a* b
/	Float Division (Quotient of two operands)	a / b
//	Floor Division (Quotient with fractional part)	a//b
%	Modulus (Integer remainder of two operands)	a % b
**	Exponent (Product of an operand n times by itself)	a ** n

Comparison Operations		
Operator	**Purpose**	**Example**
>	Greater than (If left > right hence return true)	a > b
<	Less than (if left < right hence return true)	a < b
==	Equal to (if left equals right return true)	a == b
!=	Not equal to (if left not equals right return true)	a != b
>=	Greater than or equal (if left GTE right return true)	a >= b
<=	Less than or equal (if left LE right return True)	a <= b

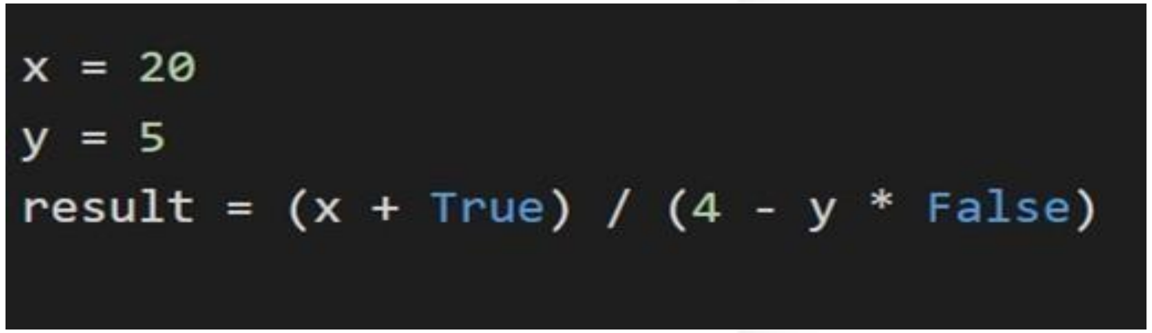
Logical		
Operator	**Purpose**	**Example**
and	If a and b are both true hence return true	a and b
or	If either a or b is true hence return true	a or b
not	If a is true return false and vice versa	not a

Identity		
Operator	**Purpose**	**Example**
is	If both operands refers to same object return True	a is b
Is not	If both operands refers to different objects return True	a is not b

Assignment Operations		
Operator	**Purpose**	**Example**
=	Assignment	10 = 2
+=	Add and assign	10 += 2
-=	Subtract and assign	10 -= 2
*=	Multiply and assign	10 *= 2
/=	Divide and assign	10 /= 2

➔What is the output of printing ‘result’?

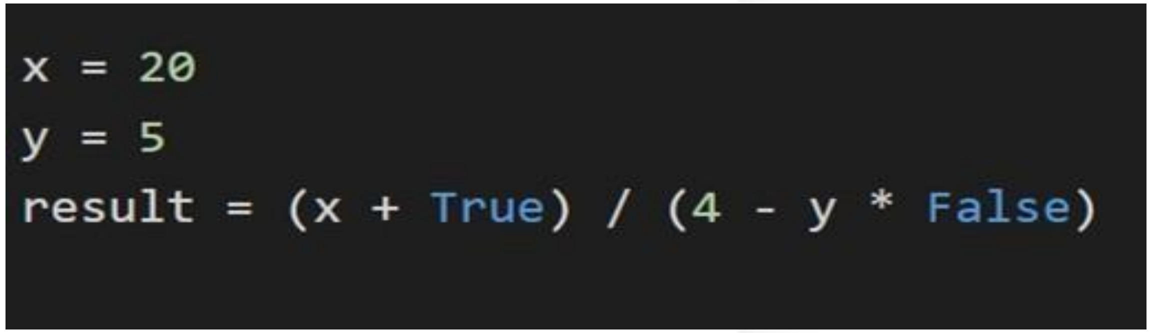
- ○ False
- ○ -21
- ○ 5
- ○ 5.25



```
x = 20
y = 5
result = (x + True) / (4 - y * False)
```

➔ What is the output of printing ‘result’?

- False
- -21
- 5
- [✓ 5.25]



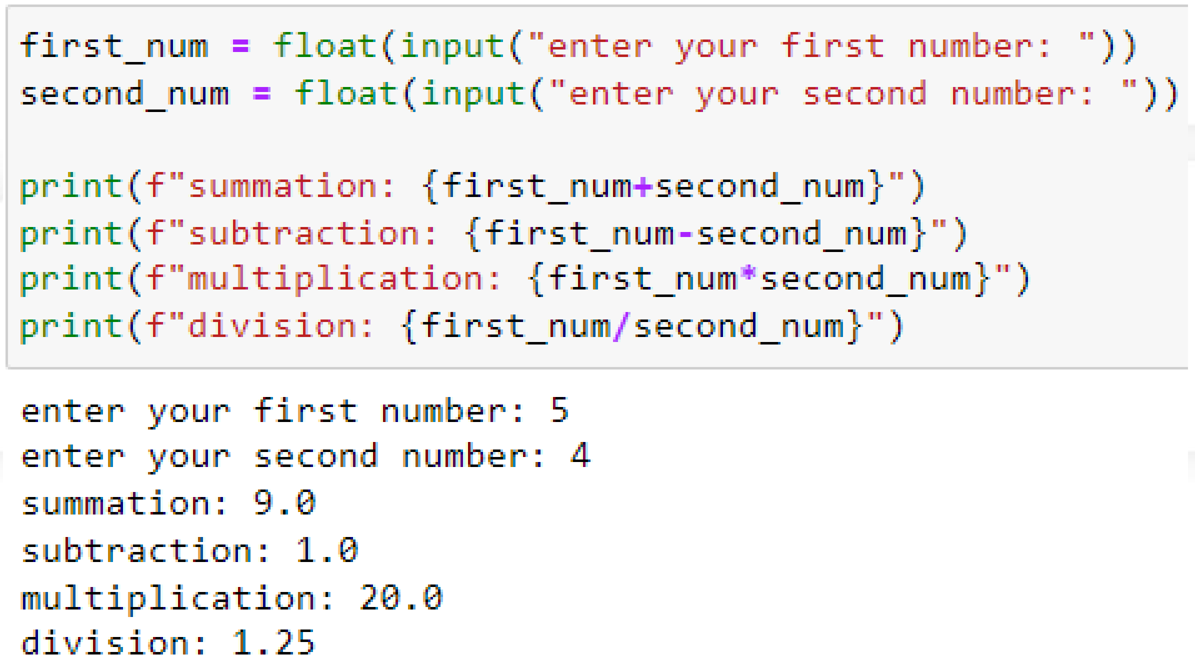
```
x = 20
y = 5
result = (x + True) / (4 - y * False)
```

Try To Solve

Write code take two numerical inputs for addition, subtraction, multiplication, and division. Your cooperation in furnishing these values will enable us to perform the specified mathematical operations accurately.

Try To Solve

Write code take two numerical inputs for addition, subtraction, multiplication, and division. Your cooperation in furnishing these values will enable us to perform the specified mathematical operations accurately.



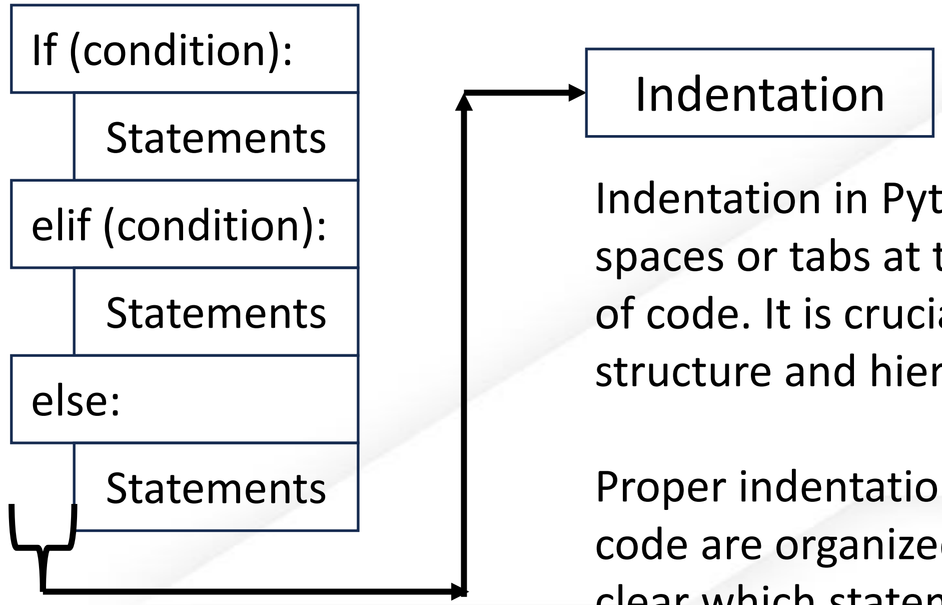
```
first_num = float(input("enter your first number: "))
second_num = float(input("enter your second number: "))

print(f"summation: {first_num+second_num}")
print(f"subtraction: {first_num-second_num}")
print(f"multiplication: {first_num*second_num}")
print(f"division: {first_num/second_num}")
```

```
enter your first number: 5
enter your second number: 4
summation: 9.0
subtraction: 1.0
multiplication: 20.0
division: 1.25
```

3- If-Condition

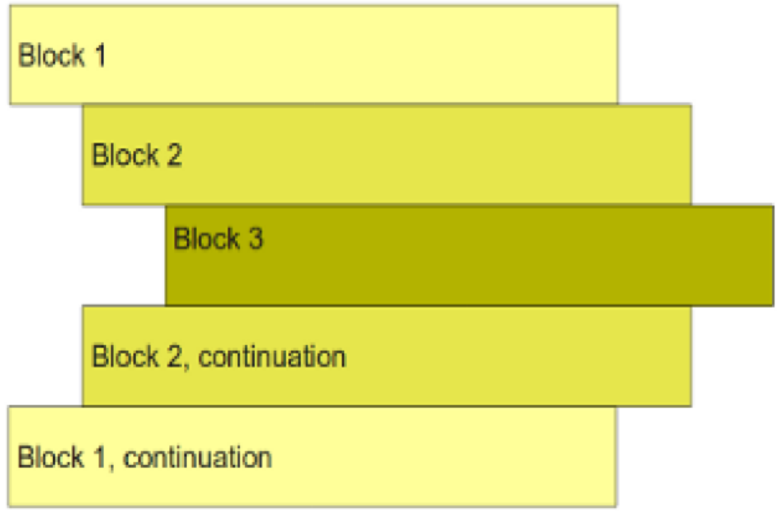
If-Condition

if condition is a control structure that allows you to execute a block of code if a specified condition is true.



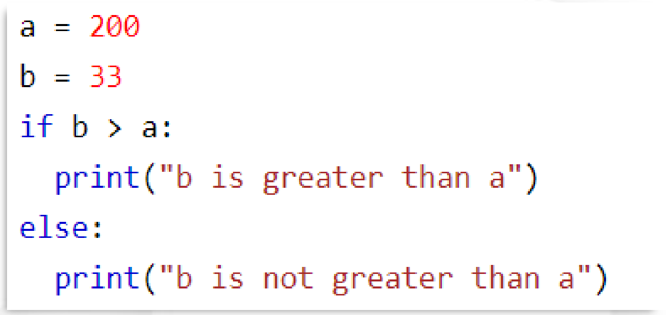
Indentation in Python refers to the spaces or tabs at the beginning of a line of code. It is crucial for indicating the structure and hierarchy of the code.

Proper indentation ensures that blocks of code are organized, making it visually clear which statements are part of a specific loop, function, or conditional statement.



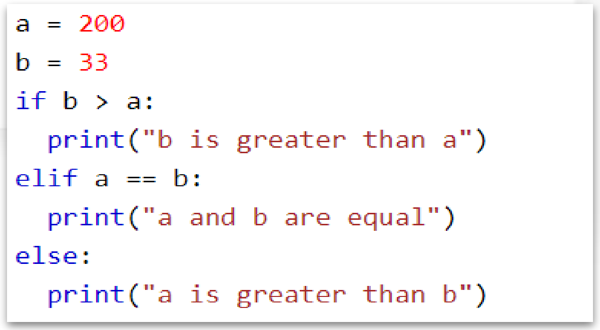
If-Condition

Example 1



```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Example 2

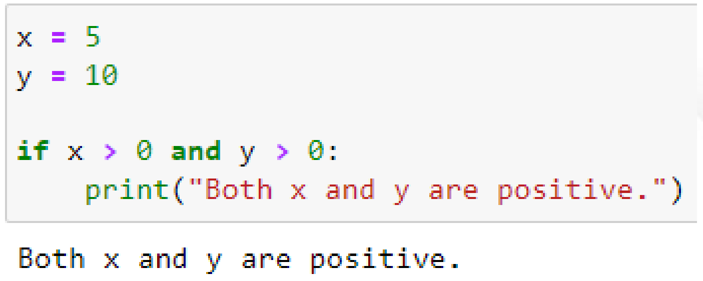


```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

AMIT

If-Condition With Logic Operations

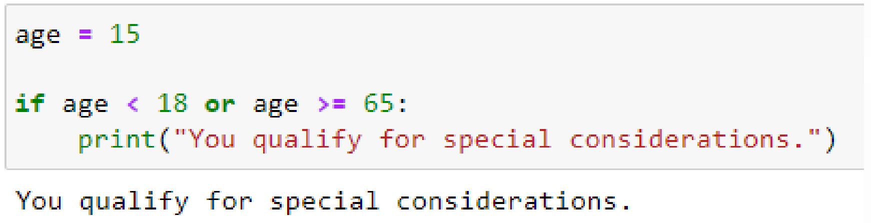
Example 1



```python
x = 5
y = 10

if x > 0 and y > 0:
    print("Both x and y are positive.")
```

```
Both x and y are positive.
```

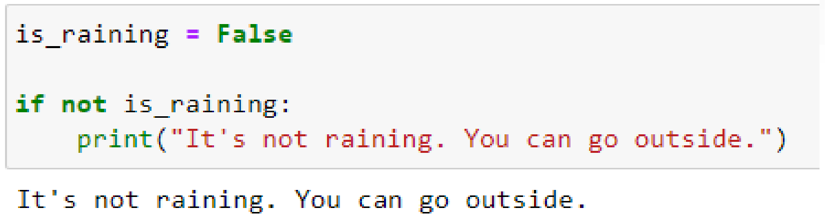
Example 2



```python
age = 15

if age < 18 or age >= 65:
    print("You qualify for special considerations.")
```

```
You qualify for special considerations.
```

Example 3

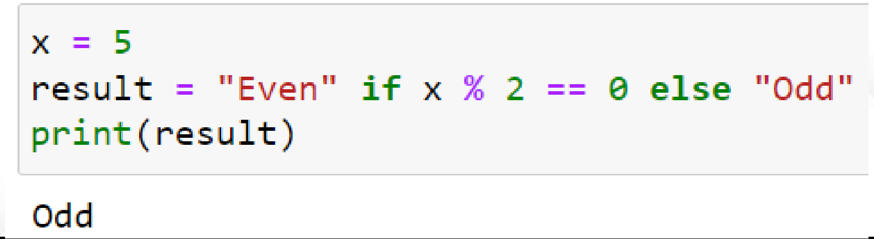


```python
is_raining = False

if not is_raining:
    print("It's not raining. You can go outside.")
```

```
It's not raining. You can go outside.
```

ShortHand If - Condition

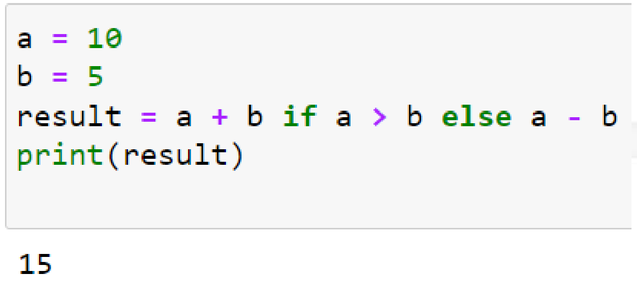
Example 1



```python
x = 5
result = "Even" if x % 2 == 0 else "Odd"
print(result)
```

```
Odd
```

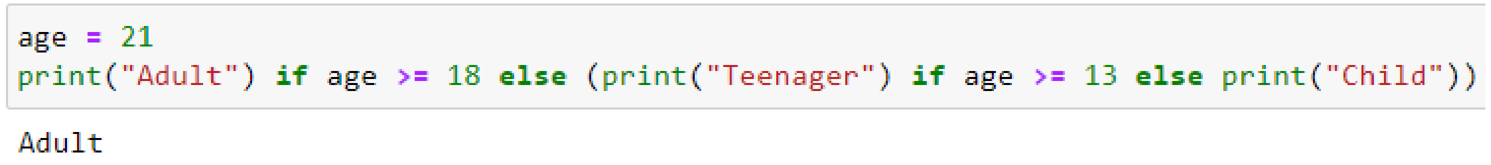
Example 2



```python
a = 10
b = 5
result = a + b if a > b else a - b
print(result)
```

```
15
```

Example 3

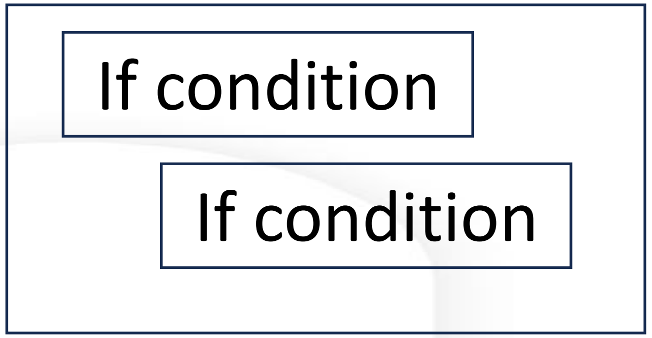


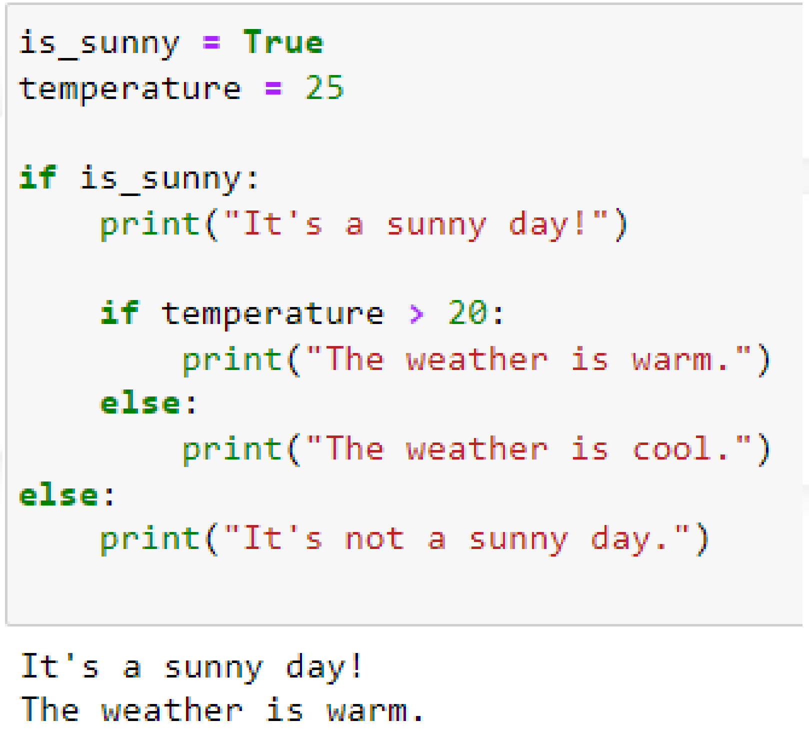
```python
age = 21
print("Adult") if age >= 18 else (print("Teenager") if age >= 13 else print("Child"))
```

```
Adult
```

Nested If - Condition

Nested if conditions refer to the use of one or more if statements inside another if statement, creating a hierarchy of conditions and code blocks.





```python
is_sunny = True
temperature = 25

if is_sunny:
    print("It's a sunny day!")

    if temperature > 20:
        print("The weather is warm.")
    else:
        print("The weather is cool.")
else:
    print("It's not a sunny day.")
```

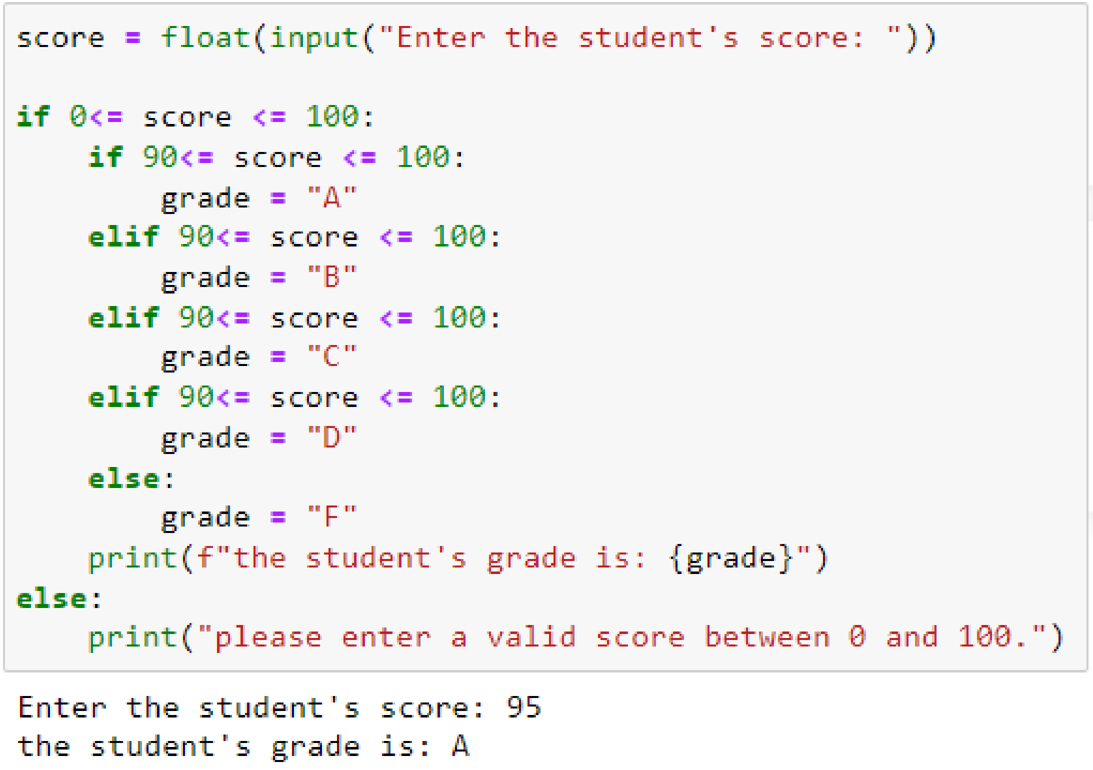
```
It's a sunny day!
The weather is warm.
```

Try To Solve

implement a Python program that calculates a student's grade based on the entered degree, where the grading system is defined as follows: 90 to 100 corresponds to 'A,' 80 to 90 corresponds to 'B,' 70 to 80 corresponds to 'C,' 60 to 70 corresponds to 'D,' and any score below 60 corresponds to 'F'?

Try To Solve

implement a Python program that calculates a student's grade based on the entered degree, where the grading system is defined as follows: 90 to 100 corresponds to 'A,' 80 to 90 corresponds to 'B,' 70 to 80 corresponds to 'C,' 60 to 70 corresponds to 'D,' and any score below 60 corresponds to 'F'?



```python
score = float(input("Enter the student's score: "))

if 0<= score <= 100:
    if 90<= score <= 100:
        grade = "A"
    elif 90<= score <= 100:
        grade = "B"
    elif 90<= score <= 100:
        grade = "C"
    elif 90<= score <= 100:
        grade = "D"
    else:
        grade = "F"
    print(f"the student's grade is: {grade}")
else:
    print("please enter a valid score between 0 and 100.")
```

```
Enter the student's score: 95
the student's grade is: A
```

4- Loops

LOOPS

Loops in Python enable the repeated execution of code, with 'for' loops iterating over sequences and 'while' loops executing as long as a specified condition holds. They streamline the handling of repetitive tasks in Python programs.

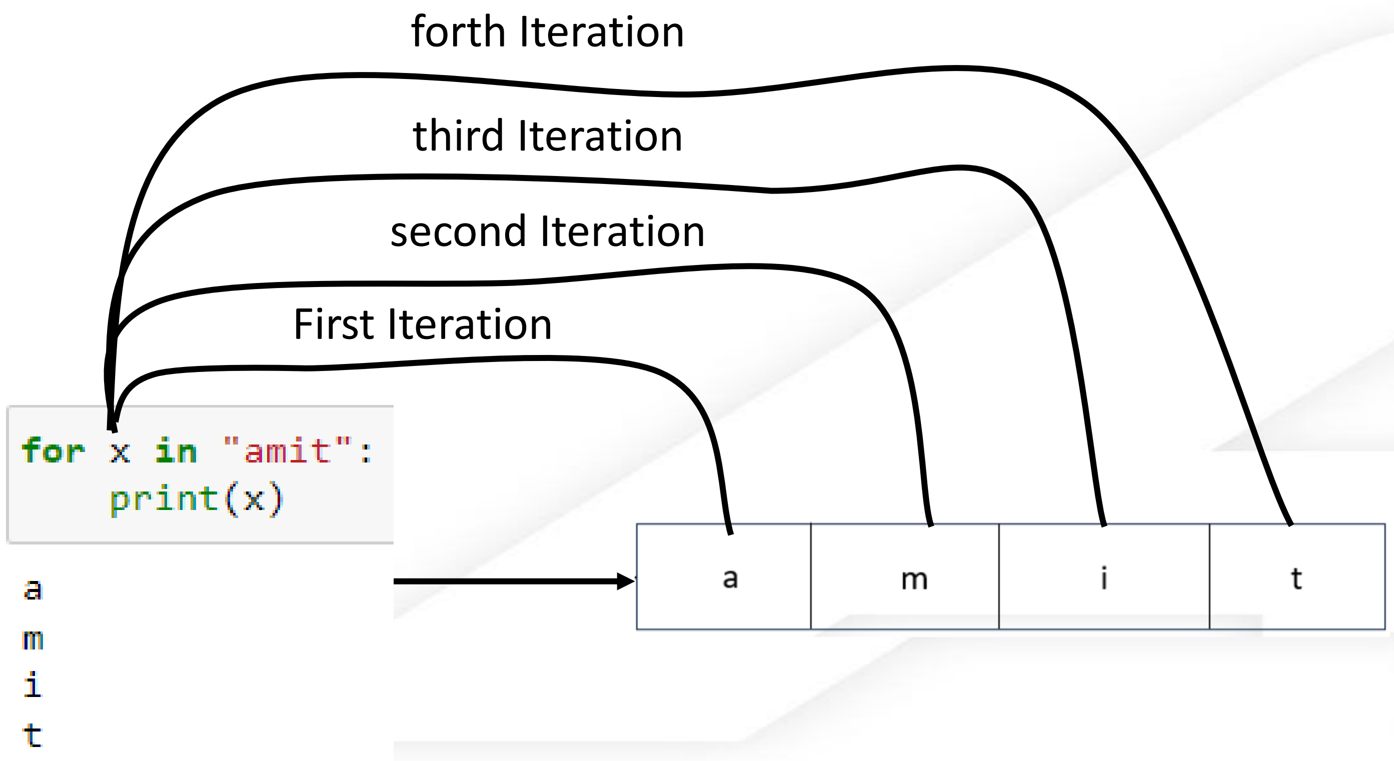
for loop

for loop in Python is a control flow statement used for iterating over a sequence (such as a list, tuple, string, or range), executing a block of code for each element in the sequence.

While loop

while loop in Python is a control flow statement that repeatedly executes a block of code as long as a specified condition remains true.

for loop



```python
for x in "amit":
    print(x)
```

```
a
m
i
t
```

Iteration 1

x = "a"

∵ x = value

∴ execute code inside for loop

Iteration 2

x = "m"

∵ x = value

∴ execute code inside for loop

Iteration 3

x = "i"

∵ x = value

∴ execute code inside for loop

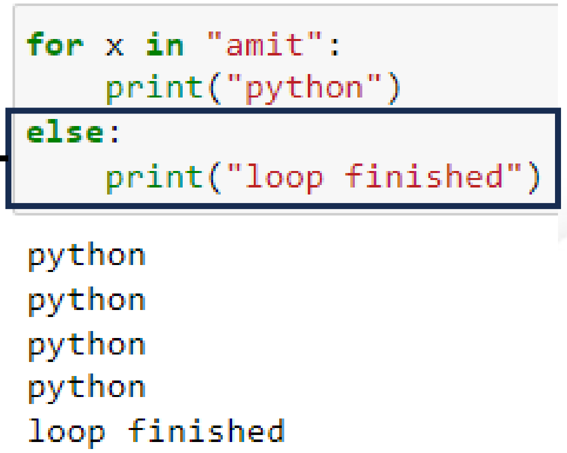
Iteration 4

x = "t"

∵ x = value

∴ execute code inside for loop

for loop



```
for x in "amit":
    print("python")
else:
    print("loop finished")
```

```
python
python
python
python
loop finished
```

Iteration 1

x = "a"

∵ x = value

∴ execute code inside for loop

Iteration 2

x = "m"

∵ x = value

∴ execute code inside for loop

Iteration 3

x = "i"

∵ x = value

∴ execute code inside for loop

Iteration 4

x = "t"

∵ x = value

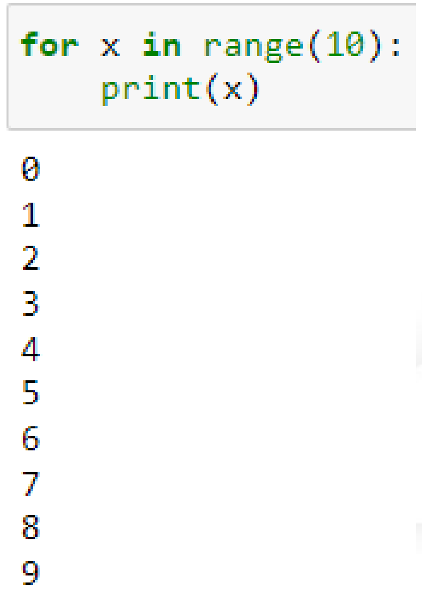
∴ execute code inside for loop

else

else block provides an opportunity to execute code after the loop has completed its iterations successfully.

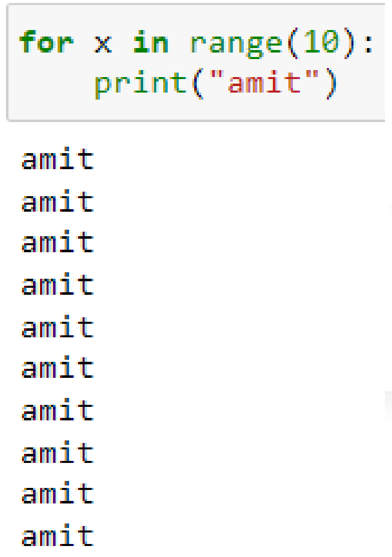
for loop

The range() function in Python generates a sequence of numbers within a specified range, providing a convenient way to iterate over a sequence of values, such as indices in a loop. It is commonly used in for loops.



```
for x in range(10):
    print(x)
```

```
0
1
2
3
4
5
6
7
8
9
```



```
for x in range(10):
    print("amit")
```

```
amit
amit
amit
amit
amit
amit
amit
amit
amit
amit
```

Iteration 1

x = 0

∵ x = value

∴ execute code inside for loop

Iteration 2

x = 1

∵ x = value

∴ execute code inside for loop

………

Iteration 10

x = 9

∵ x = value

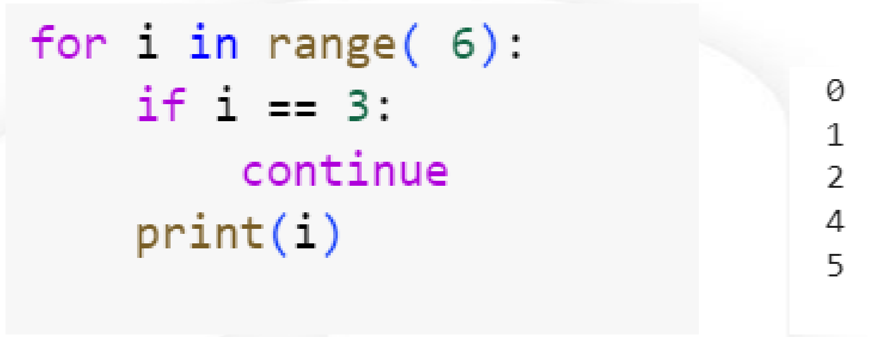
∴ execute code inside for loop

Loop Control Keywords

Loop Control Keywords		
Break	**Continue**	**pass**
break statement is used within loops to prematurely terminate the loop's execution, bypassing the remaining code inside the loop.	continue statement is used within loops to skip the rest of the code block for the current iteration and immediately move on to the next iteration of the loop.	pass statement has no effect and is often used when the interpreter expects an indented block, but there is no meaningful content to include.

Loop Control Keywords

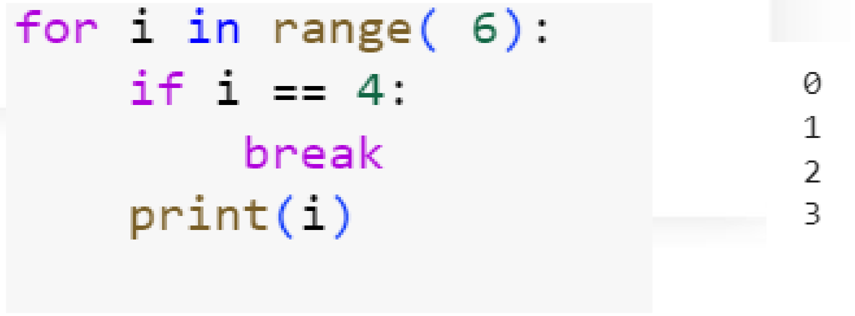
Example 1



```python
for i in range( 6):
    if i == 3:
        continue
    print(i)
```

```
0
1
2
4
5
```

Example 2

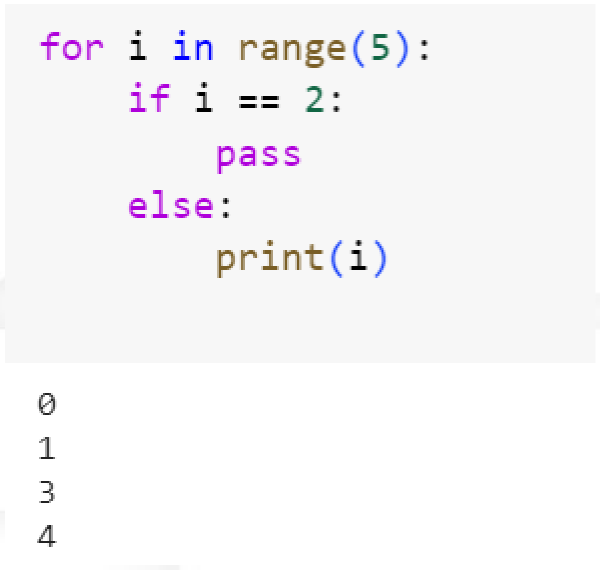


```python
for i in range( 6):
    if i == 4:
        break
    print(i)
```

```
0
1
2
3
```

Loop Control Keywords

Example 3

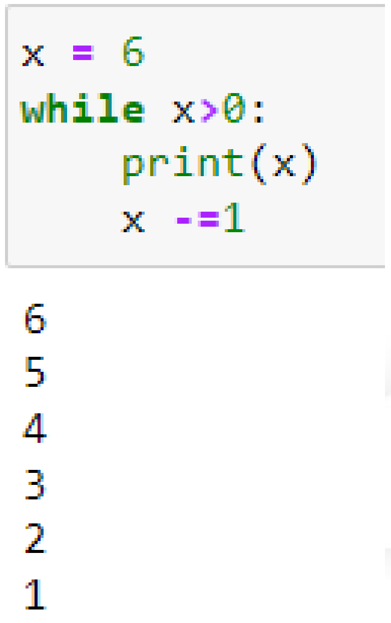


```python
for i in range(5):
    if i == 2:
        pass
    else:
        print(i)
```

```
0
1
3
4
```

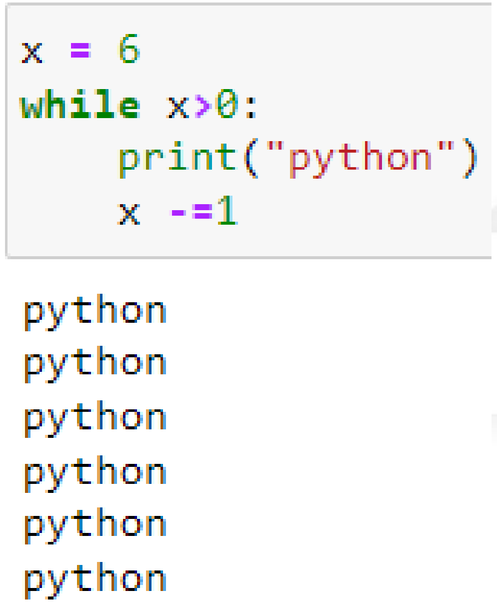
while loop

while loop in programming is a control flow structure that repeatedly executes a block of code as long as a specified condition is true, allowing for iterative execution until the condition becomes false.



```
x = 6
while x>0:
    print(x)
    x -=1
```

```
6
5
4
3
2
1
```



```
x = 6
while x>0:
    print("python")
    x -=1
```

```
python
python
python
python
python
python
```

Iteration 1

x = 6
∵ x > 0
∴ execute code inside for loop

Iteration 2

x = 5
∵ x > 0
∴ execute code inside for loop

………

Iteration 7

x = 0
∵ x not greater than 0
∴ cannot execute code inside for loop
∴ loop finished

AMIT

Thank You