Welcome Everyone!

We will wait for others to join in!

KNOW ABOUT ME:

We Will start in 10

PYTHON BASICS

With Aaryan Kapur

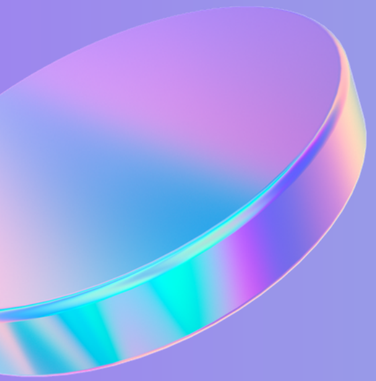
WEEK 4

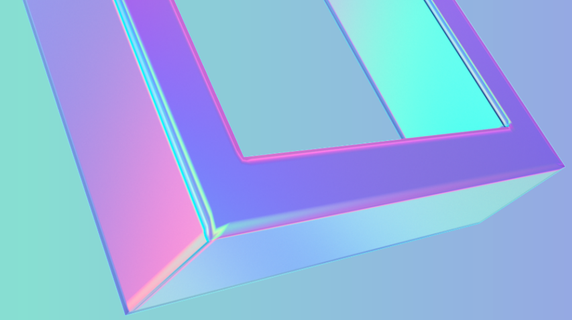
TOP PERFORMER

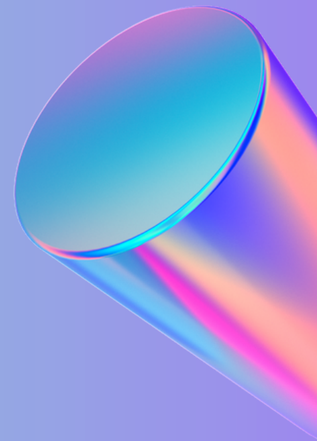
WEEK 3

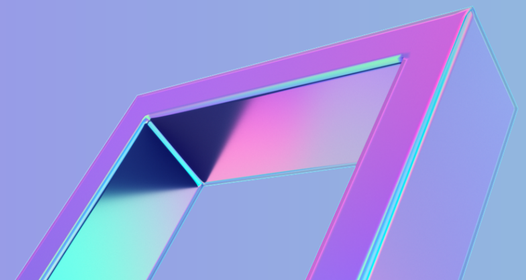
Priyam Shrivastava

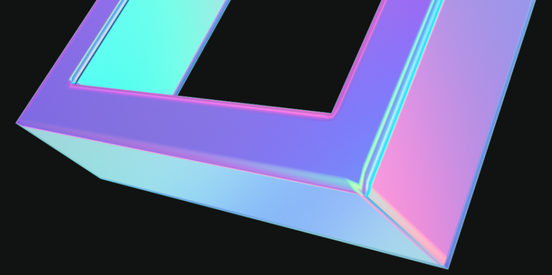
LOOPS







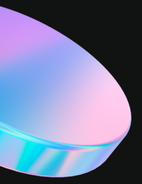


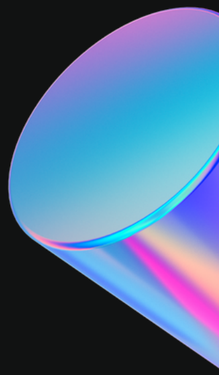


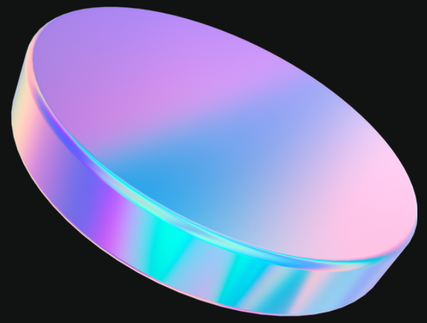
Why do we use loops?

ITERATIONS & REPETITIVE WORK

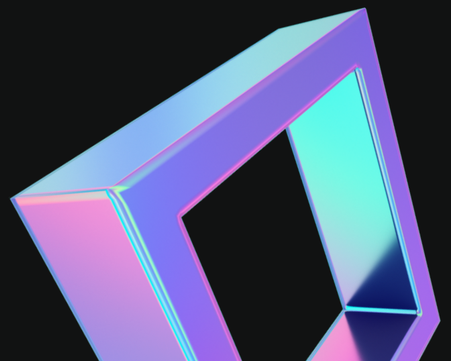
Iterative actions spanning multiple items in a structure might take a long time if done manually! loops make life easier by taking care of iterative work on providing necessary parameters.







Types of loops



- ## While Loop

  As long as a condition is true

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

- ## For Loop

  Once for each item in
  a list, tuple, set, range, etc..

```
for x in range(1,6):
print(x)
```

```
1
2
3
4
5
```

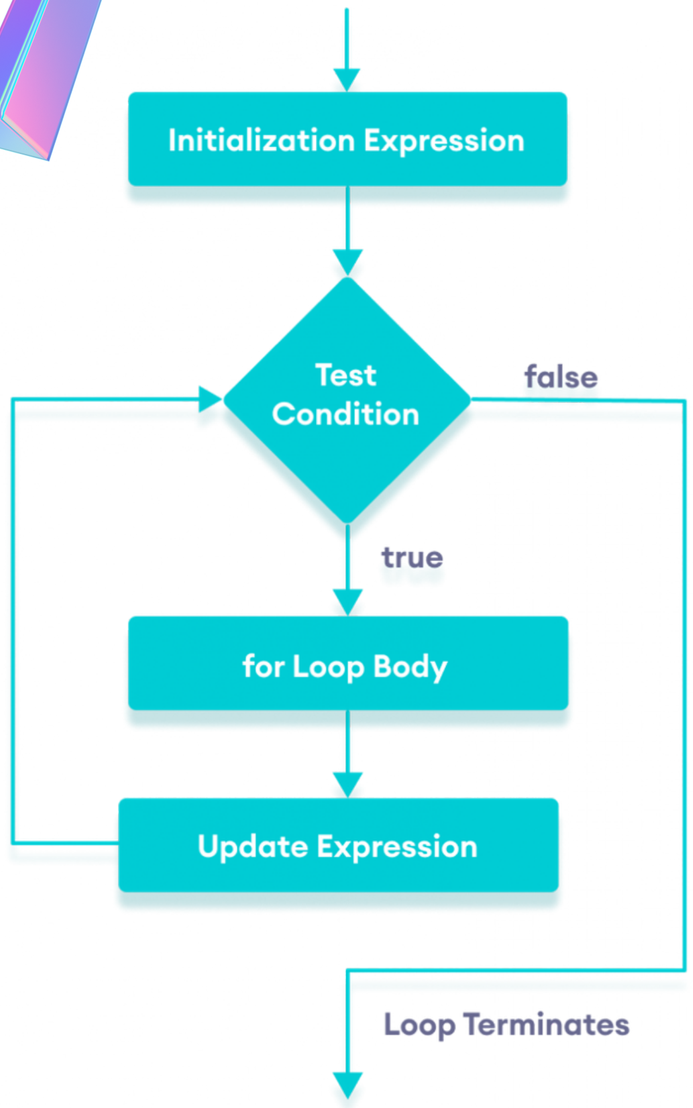
While Loop

While loops will execute as long as a condition is satisfied

```
i = 1
while(i<=10):
  print("Attempt Number:", i)
  i += 1
```

Syntax:

```
while(Condition):
    code to run
```



For Loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
for i in range(1,11):
  print("Attempt Number:", i)
```

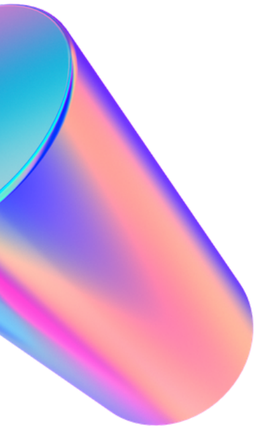
Syntax:

```
for i in Condition:
    code to run
```

NESTED LOOPS

You can use Loops inside of Loops

These enable conditions to be put in conditions that helps us make even patterns and other useful things such as accessing values in values!



```
for i in range(1,11):
    for j in range(11,21):
        print("i = ",i,"& j =",j)
```

```
for i in range(1,11):
    j = 10
    while(j<21):
        print("i = ",i,"& j =",j)
        j+=1
```

Control Statements

Continue Statement

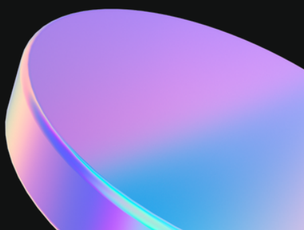
Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

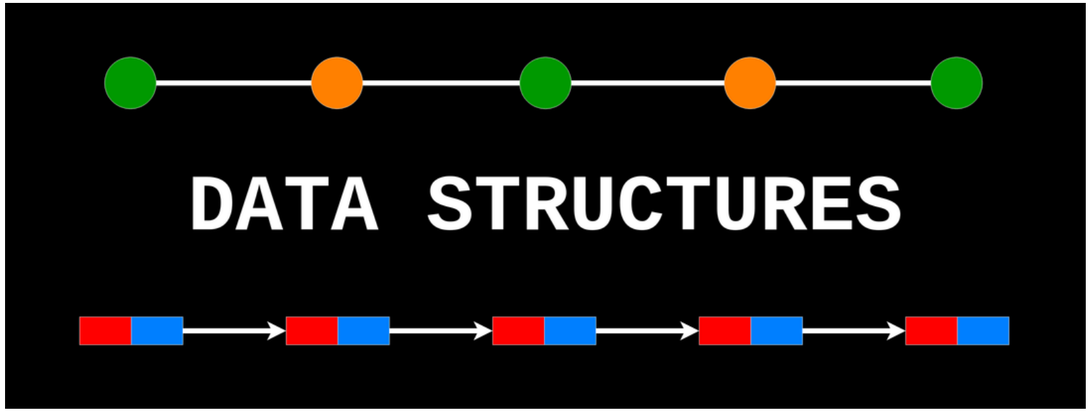
Pass Statement

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Break Statement

Terminates the loop statement and transfers execution to the statement immediately following the loop.





DATA STRUCTURES

Multiple values of data denoted in a cumulative manner!

There are several structures of data in Python:

List is a collection that is ordered and changeable. Allows duplicate members.

Tuple is a collection that is ordered and unchangeable. Allows duplicate members.

Set is a collection that is unordered and unindexed. No duplicate members.

Dictionary is a collection that is ordered* and changeable. No duplicate members.

Default Data Structures in Python

LISTS IN PYTHON

```
list1= ["abc", 34, True, 40, "male"]
```

TUPLES IN PYTHON

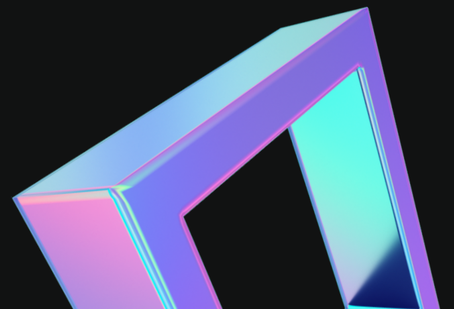
```
tuple1 = ("apple", 1,"banana", "cherry")
```

SETS IN PYTHON

```
set1= {"apple", "banana", "cherry"}
```

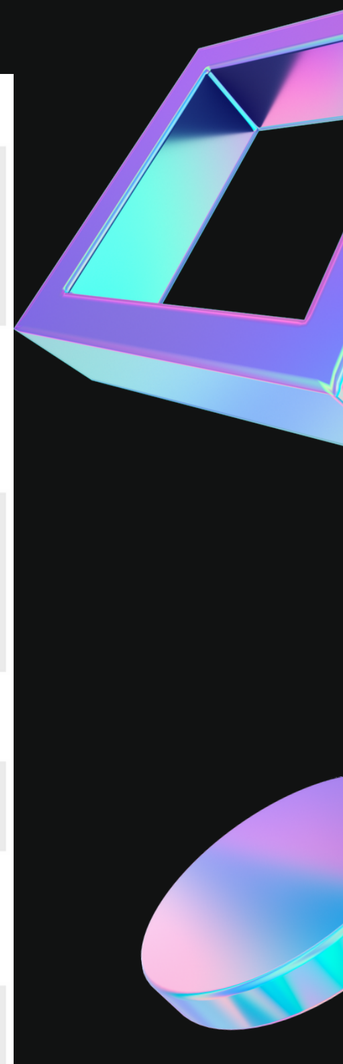
DICTIONARY IN PYTHON

```
dict1 = {"brand": "Ford",
"model": "Mustang", "year": 1964}
```



Difference Between Data Types

List	Tuple	Set	Dictionary
List is a non-homogeneous data structure which stores the elements in single row and multiple rows and columns	Tuple is also a non-homogeneous data structure which stores single row and multiple rows and columns	Set data structure is also non-homogeneous data structure but stores in single row	Dictionary is also a non-homogeneous data structure which stores key value pairs
List can be represented by [ ]	Tuple can be represented by ( )	Set can be represented by { }	Dictionary can be represented by { }
List allows duplicate elements	Tuple allows duplicate elements	Set will not allow duplicate elements	Set will not allow duplicate elements but keys are not duplicated
List can use nested among all	Tuple can use nested among all	Set can use nested among all	Dictonary can use nested among all
Example: [1, 2, 3, 4, 5]	Example: (1, 2, 3, 4, 5)	Example: {1, 2, 3, 4, 5}	Example: {1, 2, 3, 4, 5}
List can be created using **list()** function	Tuple can be created using **tuple()** function.	Set can be created using **set()** function	Dictonary can be created using **dict()** function.
List is mutable i.e we can make any changes in list.	Tuple is immutable i.e we can not make any changes in tuple	Set is mutable i.e we can make any changes in set. But elements are not duplicated.	Dictionary is mutable. But Keys are not duplicated.
List is ordered	Tuple is ordered	Set is unordered	Dictionary is ordered



LISTS

list1= ["abc", 34, True, 40, "male"]

- Python lists do the work of most of the collection data structures found in other languages.
- Lists can be used for any type of object, from numbers and strings to more lists.
- They are accessed just like strings (e.g. slicing and concatenation) so they are simple to use and they’re variable length, i.e. they grow and shrink automatically as they’re used.

Lists!

list1= ["a", 3, 4]
list2= [66, 3, 4]

Add items to list

```
list1.append(1)
print(list1)
['a', 3, 4, 1]
```

Insert items in list

```
list1.insert(2,"hello")
print(list1)
['a', 3, 'hello', 4]
```

Count item in list

```
a = list2.count(3)
print(a)
1
```

Remove items from list

```
list1.remove(3)
print(list1)
['a', 4]
```

Pop items from list

```
list1.pop(2)
print(list1)
['a', 3]
```

Reverse a list

```
list2.reverse()
print(list2)
[4, 3, 66]
```

Sort list in ascending order

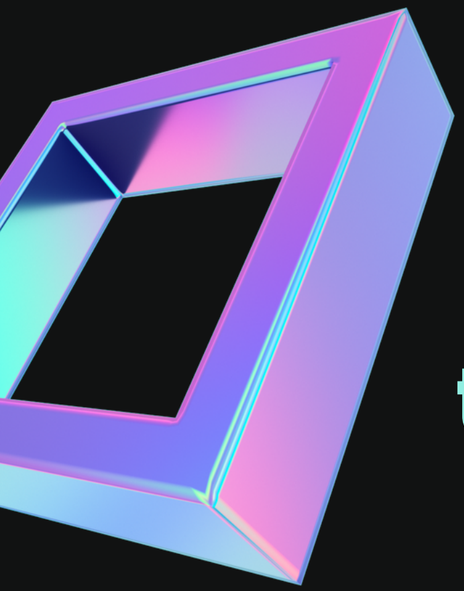
```
list2.sort()
print(list2)
[3, 4, 66]
```

Sort list in ascending order

```
list2.sort(reverse = True)
print(list2)
[66, 4, 3]
```

Find index of item

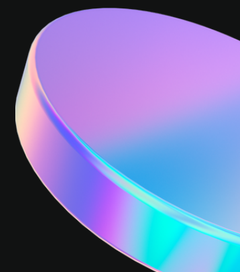
```
a = list2.index(3)
print(a)
1
```



Tuples

```
tuple1 = ("apple", 1,"banana", "cherry")
```

- Tuples are used to store multiple items in a single variable.
- Tuples are written with round brackets.
- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.



Tuples!

tuple1 = (1,3,"a")
tuple2 = (12,34,5)

Add items to tuple

```
tuple1.append(1)
print(tuple1)
```

Error

Insert items in tuple

```
tuple1.insert(2,"hello")
print(tuple1)
```

Error

Count item in tuple

```
a = tuple1.count(3)
print(a)
```

1

Remove items from tuple

```
tuple1.remove(1)
print(tuple1)
```

Error

Pop items from tuple

```
tuple1.pop(1)
print(tuple1)
```

Error

Reverse a tuple

```
tuple1.reverse()
print(tuple1)
```

Error

Sort tuple

```
tuple1.sort()
print(tuple1)
```

Error

Delete a complete Tuple

```
del tuple2
print(tuple2)
```

Error

Find index of item

```
a = tuple1.index(3)
print(a)
```

1

Sets

set1= {"apple", "banana", "cherry"}

- Sets are used to store multiple items in a single variable.
- Sets are written with curly brackets.
- Set items are unordered, unchangeable.
- Duplicate values are not ordered

Sets!

set1 = {1,"s",3}
set2 = {20,3,33}

Add items to set

```
set2.add(22)
print(set2)
{33, 20, 3, 22}
```

Remove items from set

```
set2.remove(20)
print(set2)
{33, 4}
```

Find if set1 is subset of set2

```
a = set1.issubset(set2)
print(a)
False
```

Difference between 2 sets

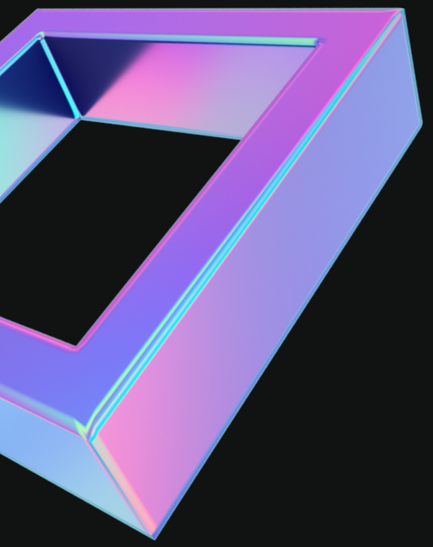
```
a = set1.difference(set2)
print(a)
{1, 's'}
```

Intersection between 2 sets

```
a = set1.intersection(set2)
print(a)
{3}
```

Find if set1 is superset of set2

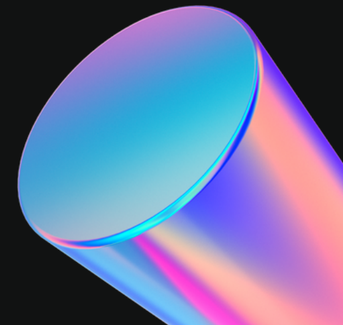
```
a = set1.issuperset(set2)
print(a)
False
```



Dictionary

```
dict1 = {"brand": "Ford",
"model": "Mustang", "year": 1964}
```

- Dictionaries are used to store data values in key: value pairs.
- A dictionary is a collection that is ordered*, changeable, and does not allow duplicates.
- Dictionaries are written with curly brackets and have keys and values.
- Dictionary items are presented in key: value pairs, and can be referred to by using the key name.



Dictionary!

dict1 = {1:"One", 2: "Two",3: "Three"}
dict2 = {3:"Three", 4: "Four",5: "Five"}

Print all Keys

```
a = dict1.keys()
print(a)
dict_keys([1, 2, 3]
```

Print all Values

```
a = dict1.values()
print(a)
dict_values(['One', 'Two', 'Three'])
```

Adding Values

```
dict1[4] = "Four"
print(dict1)
{1: 'One', 2: 'Two', 3: 'Three', 4: 'Four'}
```

Empty Dict

```
a = dict1.clear()
print(dict1)
{}
```

Update Values

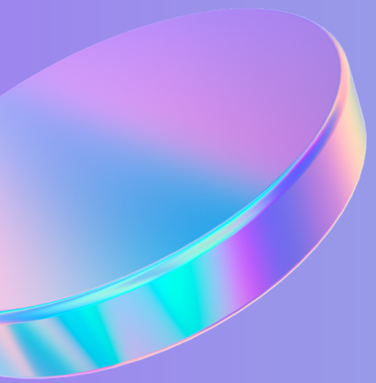
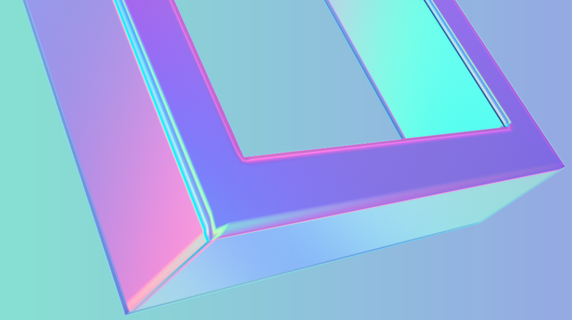
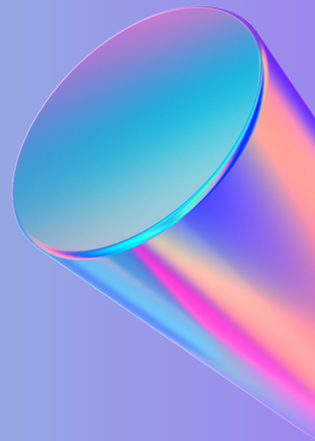
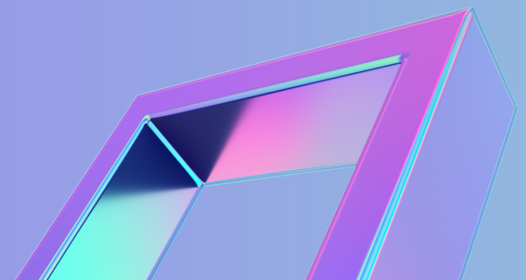
```
dict1.update({3:"TRI"})
print(dict1)
{1: 'One', 2: 'Two', 3: 'TRI'}
```

Delete Item

```
a = dict1.pop(2)
print(dict1)
{1: 'One', 3: 'Three'}
```

Nested Data Structures

WE WILL CONTINUE THIS IN THE NEXT SESSION!

HANDS ON TIME!

RACE TO BE THE TOP PERFORMER!

Thank You

WEBSITE

aaryankapur.tech