**Dictionary, Tuple, Set, If-Else Condition, While Loop, For Loop**

**Dictionary**

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.List is a mutable data type. I.e. value of the dictionary can be altered or modified.

Example  
Create and print a dictionary:  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
print(thisdict)

Output:{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

**Accessing Items**

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example  
Get the value of the "model" key:

x = thisdict["model"]

print x

Output:Mustang

**Change Values**

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict["year"] = 2018

Output:{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}

**Loop Through a Dictionary**

You can loop through a dictionary by using a for loop. When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

Print **keys & corresponding** values in dictionary

Example:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

for x in thisdict:

print (x, thisdict[x])

OR

for k,v in thisdict.items():

print (k,v)

Example

Print all **key names** in the dictionary, one by one:

for x in thisdict:

print(x)

Output:

brand

model

year

Example

Print all **values** in the dictionary, one by one:

for x in thisdict:

print(thisdict[x])  
Output:

Ford

Mustang

1964

**Dictionary Length**

To determine how many items (key-value pairs) a dictionary has, use the **len()** method.

Example

Print the number of items in the dictionary:

print(len(thisdict))

Output:3

**Adding Items**

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict["color"] = "red"

print(thisdict)

Output:{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

**Removing Items**

The **del** keyword removes the item with the specified key name

Example

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

del thisdict["model"]

print(thisdict)

Output:{'brand': 'Ford', 'year': 1964}

**Making dictionary empty**

The **clear()** keyword empties the dictionary

Example

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict.clear()

print(thisdict)

Output:{}

**Deleting the dictionary completely**

The **del** keyword can also delete the dictionary completely

Example

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

del thisdict

Output:<type 'dict'>

**Tuple**

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets. Tuple is a immutable data type i.e. value of the tupple cannot be changed or modified

Example

Create a Tuple:

thistuple = ("apple", "banana", "cherry")

print(thistuple)

Output:('apple', 'banana', 'cherry')

**Access Tuple Items**

You can access tuple items by referring to the index number, inside square brackets:

Example

Return the item in position 1:

thistuple = ("apple", "banana", "cherry")

print(thistuple[1])

Output:banana

**Change Tuple Values**

Once a tuple is created, you cannot change its values. Tuples are unchangeable.

Example

You cannot change values in a tuple:

An error message is shown if I try to change value in the tuple

thistuple = ("apple", "banana", "cherry")

thistuple[1] = "blackcurrant"

print(thistuple)

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

**Loop Through a Tuple**

You can loop through the tuple items by using a for loop.

Example

Iterate through the items and print the values:

thistuple = ("apple", "banana", "cherry")

for x in thistuple:

print(x)

Output:

apple

banana

cherry

**Tuple Length**

To determine how many items a tuple has, use the **len()** method:

Example

Print the number of items in the tuple:

thistuple = ("apple", "banana", "cherry")

print(len(thistuple))

Output:3

**Add Items**

Once a tuple is created, you cannot add items to it. Tuples are unchangeable.

Example

You cannot add items to a tuple:

An error message is shown if I try to add item in tuple

thistuple = ("apple", "banana", "cherry")

thistuple[3] = "orange" # This will raise an error

print(thistuple)

Output:'tuple' object does not support item assignment

**Remove Items**

You cannot remove items in a tuple. Tuples are unchangeable, so you cannot remove items from it.

Example

You cannot remove items in a tuple.

An error message is shown if I try to remove item from tuple

thistuple = ("apple", "banana", "cherry")

del thistuple[0] # This will raise an error

'tuple' object doesn't support item deletion

**Deleting the tuple completely**

Example

The del keyword can delete the tuple completely:

thistuple = ("apple", "banana", "cherry")

del thistuple

print(thistuple)]

Output:NameError: name 'thistuple' is not defined

**Set**

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Example

Create a Set:

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

print(thisset)

Note: Sets are unordered, so the items will appear in a random order.

Output:{'banana', 'cherry', 'apple'}

**Access Items**

You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the **in** keyword.

Example

Loop through the set, and print the values:

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

for x in thisset: print(x)

Output:

banana

cherry

apple

**Change Items**

Once a set is created, you cannot change its items, but you can add new items.

Example

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

thisset[0] = "mango"

Output: 'set' object does not support item assignment

**Add Items**

To add one item to a set use the **add()** method.

To add more than one item to a set use the **update()** method.

Example

Add an item to a set, using the add() method:

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

thisset.add("orange")

print(thisset)

Output:{'apple', 'banana', 'orange', 'cherry'}

Add multiple items to a set, using the **update()** method:

Example

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

thisset.update(["orange", "mango", "grapes"])

print(thisset)

Output:{'banana', 'grapes', 'cherry', 'mango', 'orange', 'apple'}

**Get the Length of a Set**

To determine how many items a set has, use the **len()** method.

Example

Get the number of items in a set:

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

print(len(thisset))

Output:3

**Remove Item**

To remove an item in a set, use the **remove()**, or the **discard()** method.

Example

Remove "banana" by using the remove() method:

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

thisset.remove("banana")

print(thisset)

Note: If the item to remove does not exist, remove() will raise an error.

Output:{'apple', 'cherry'}

**Making a set empty**

Example

The **clear()** method empties the set:

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

thisset.clear()

print(thisset)

Output:set([])

**Delete the set completely**

Example

The **del** keyword will delete the set completely:

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

del thisset

print(thisset)

Output:NameError: name 'thisset' is not defined

**Loop** - In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times. A loop statement allows us to execute a statement or group of statements multiple times.

**If-Else Condition**

Python supports the usual logical conditions from mathematics:

* Equals: a == b
* Not Equals: a != b
* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the **if** keyword.

Example

If statement:

a = 33

b = 200

if b > a:

print("b is greater than a")

Output:b is greater than a

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

**Indentation**Python relies on indentation, using whitespace, to define scope in the code. Other programming languages often use curly-brackets for this purpose.  
Example  
If statement, without indentation (will raise an error):  
a = 33  
b = 200  
if b > a:  
print("b is greater than a") # you will get an error  
Output:File "demo\_if\_error.py", line 4  
print("b is greater than a")  
^  
IndentationError: expected an indented block

**Else**

The **else** keyword catches anything which isn't caught by the preceding conditions.

Example

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")

Output:a is greater than b

In this example a is greater to b, so the first condition is not true, also the elif condition is not true, so we go to the elsecondition and print to screen that "a is greater than b".

**And**

The **and** keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

if a > b and c > a:

print("Both conditions are True")

Output:Both conditions are True

**Or**

The **or** keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c:

if a > b or a > c:

print("At least one of the conditions is True")

Output:At least one of the conditions is True

**While loop**

With the **while** loop we can execute a set of statements as long as a condition is true.

Example

Print i as long as i is less than 6:

i = 1

while i < 6:

print(i)

i += 1

Output:

1

2

3

4

5

Note: remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

**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).

This is less like the for keyword in other programming language, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list:

fruits = ["apple", "banana", "cherry"]

for x in fruits:

print(x)

Output:

apple

banana

Cherry

**Looping Through a String**

Even strings are iterable objects, they contain a sequence of characters:

Example

Loop through the letters in the word "banana":

for x in "banana":

print(x)

Output:b

a

n

a

n

a

**The range() Function**

To loop through a set of code a specified number of times, we can use the **range()** function,

The **range()** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example

Using the range() function:

for x in range(6):

print(x)

Output:

0

1

2

3

4

5

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The **range()** function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

Example

Using the start parameter:

for x in range(2, 6):

print(x)

Output:

2

3

4

5