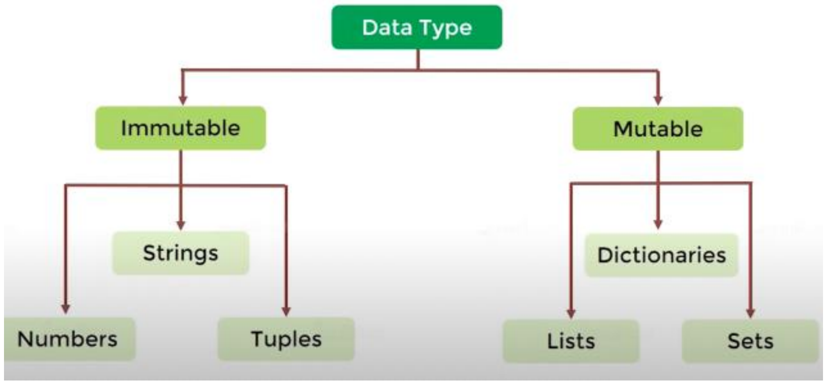
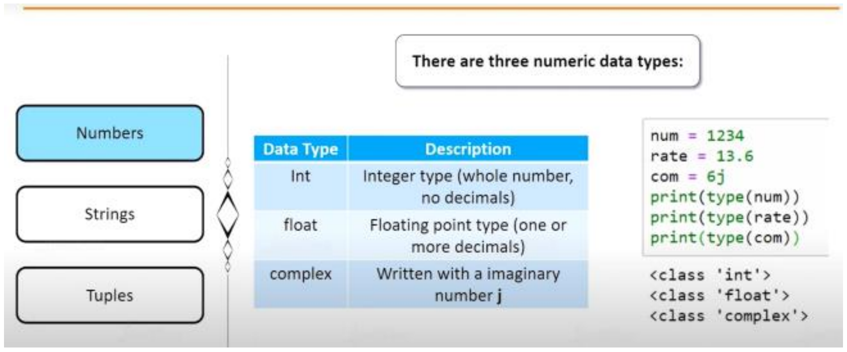
DATA TYPES and also for loop

Thursday, August 18, 2022 10:51 AM

A



Numbers


Numbers
Strings
Tuples

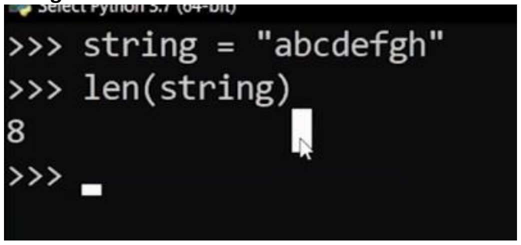
There are three numeric data types:

Data Type	Description
Int	Integer type (whole number, no decimals)
float	Floating point type (one or more decimals)
complex	Written with a imaginary number **j**

```
num = 1234
rate = 13.6
com = 6j
print(type(num))
print(type(rate))
print(type(com))
```

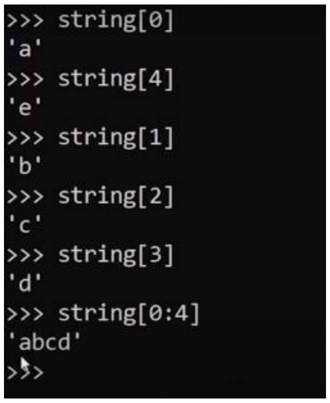
```
<class 'int'>
<class 'float'>
<class 'complex'>
```

String methods



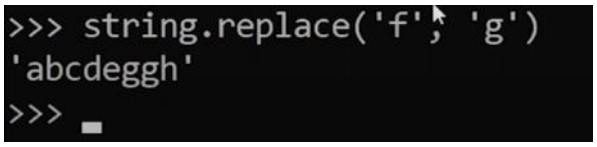
```
>>> string = "abcdefgh"
>>> len(string)
8
>>>
```

We can find the value by the index number



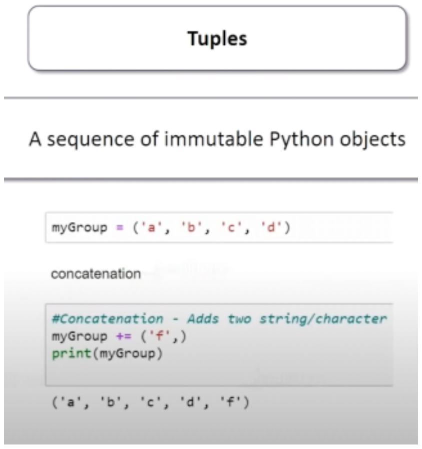
```
>>> string[0]
'a'
>>> string[4]
'e'
>>> string[1]
'b'
>>> string[2]
'c'
>>> string[3]
'd'
>>> string[0:4]
'abcd'
>>>
```

In the above array we can also perform the replace



```
>>> string.replace('f', 'g')
'abcdeggh'
>>>
```

Tuples



Tuples

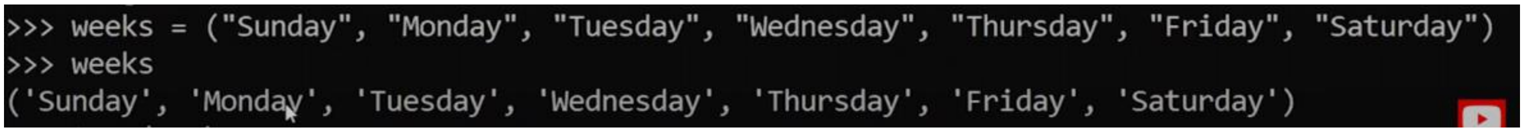
A sequence of immutable Python objects

```
myGroup = ('a', 'b', 'c', 'd')
```

concatenation

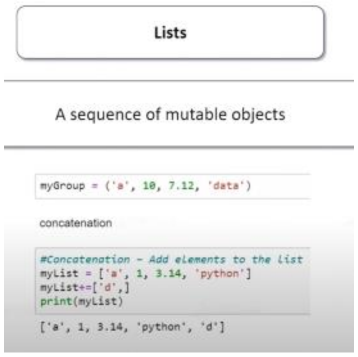
```
#Concatenation - Adds two string/character
myGroup += ('f',)
print(myGroup)
```

```
('a', 'b', 'c', 'd', 'f')
```



```
>>> weeks = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")
>>> weeks
('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday')
```

LIST



Lists

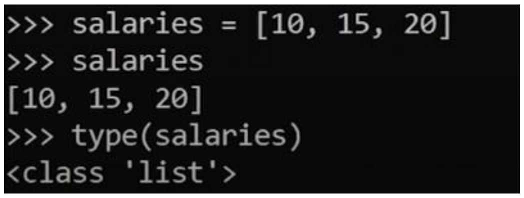
A sequence of mutable objects

```
myGroup = ('a', 10, 7.12, 'data')
```

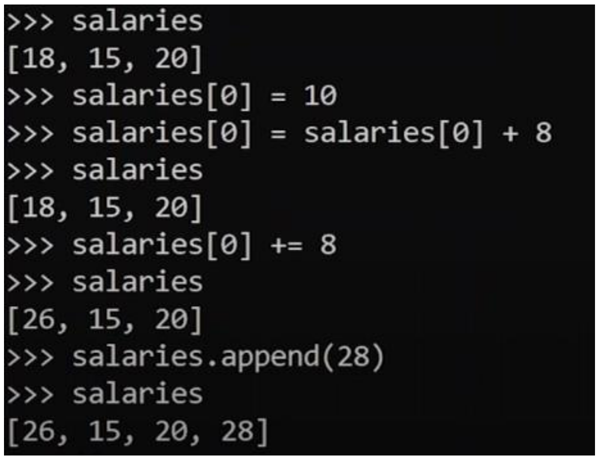
concatenation

```
#Concatenation - Add elements to the list
myList = ['a', 1, 3.14, 'python']
myList+=['d',]
print(myList)
```

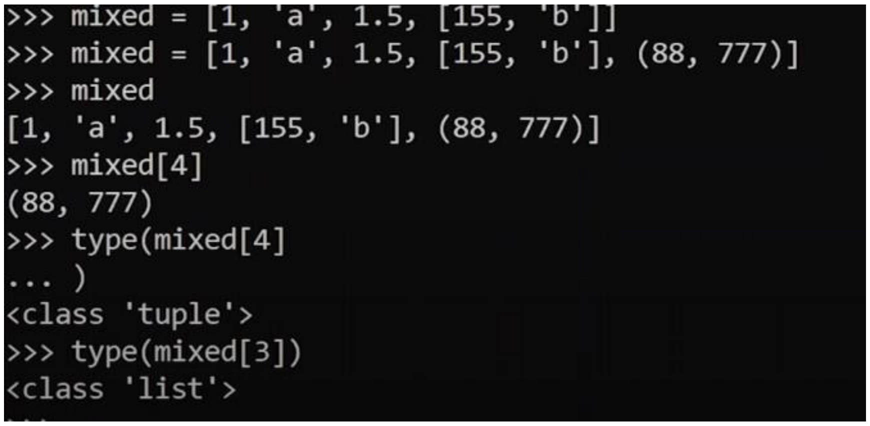
```
['a', 1, 3.14, 'python', 'd']
```



```
>>> salaries = [10, 15, 20]
>>> salaries
[10, 15, 20]
>>> type(salaries)
<class 'list'>
```

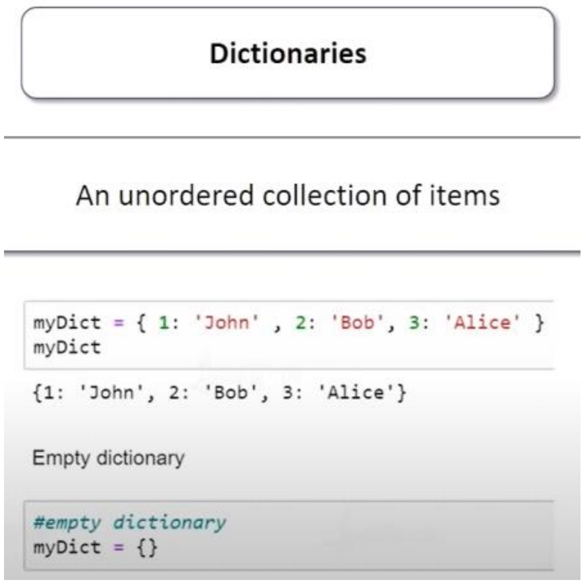

```
>>> salaries
[18, 15, 20]
>>> salaries[0] = 10
>>> salaries[0] = salaries[0] + 8
>>> salaries
[18, 15, 20]
>>> salaries[0] += 8
>>> salaries
[26, 15, 20]
>>> salaries.append(28)
>>> salaries
[26, 15, 20, 28]
```

For tuple we use () and for list we use []


```
>>> mixed = [1, 'a', 1.5, [155, 'b']]
>>> mixed = [1, 'a', 1.5, [155, 'b'], (88, 777)]
>>> mixed
[1, 'a', 1.5, [155, 'b'], (88, 777)]
>>> mixed[4]
(88, 777)
>>> type(mixed[4]
... )
<class 'tuple'>
>>> type(mixed[3])
<class 'list'>
```

The above image justify that in a list we can save the tuple as you can see we saved a int string and float with the list in it and a tuple in it

Dictionaries

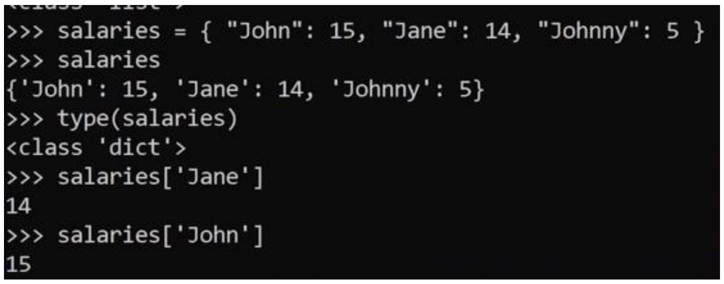

Dictionaries

An unordered collection of items

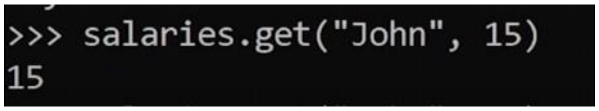
```
myDict = { 1: 'John' , 2: 'Bob', 3: 'Alice' }
myDict
```
```
{1: 'John', 2: 'Bob', 3: 'Alice'}
```

Empty dictionary

```
#empty dictionary
myDict = {}
```

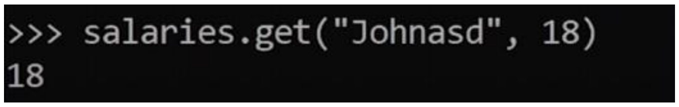


```
>>> salaries = { "John": 15, "Jane": 14, "Johnny": 5 }
>>> salaries
{'John': 15, 'Jane': 14, 'Johnny': 5}
>>> type(salaries)
<class 'dict'>
>>> salaries['Jane']
14
>>> salaries['John']
15
```



```
>>> salaries.get("John", 15)
15
```

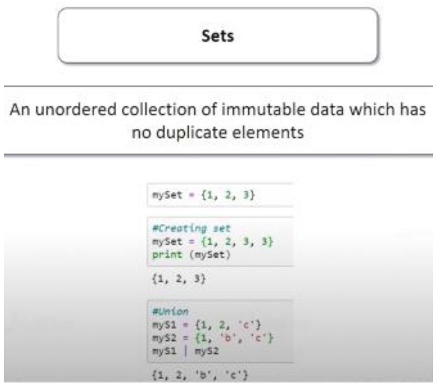
Here we are using get method where I want value of john if john is there in dictionary it will return john value or if there is no john it will give 15



```
>>> salaries.get("Johnasd", 18)
18
```

As you can see we do not have Johnasd so we get 18

Sets



Sets

An unordered collection of immutable data which has no duplicate elements

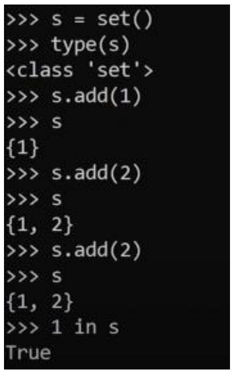
```
mySet = {1, 2, 3}
```

```
#Creating set
mySet = {1, 2, 3, 3}
print (mySet)
```

```
{1, 2, 3}
```

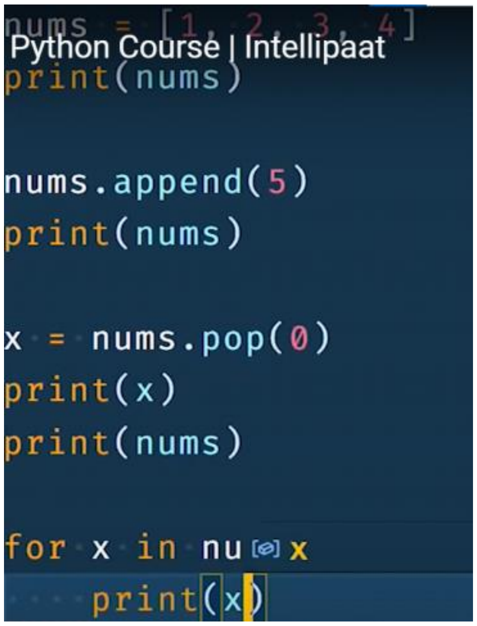
```
#Union
myS1 = {1, 2, 'c'}
myS2 = {1, 'b', 'c'}
myS1 | myS2
```

```
{1, 2, 'b', 'c'}
```



```
>>> s = set()
>>> type(s)
<class 'set'>
>>> s.add(1)
>>> s
{1}
>>> s.add(2)
>>> s
{1, 2}
>>> s.add(2)
>>> s
{1, 2}
>>> 1 in s
True
```

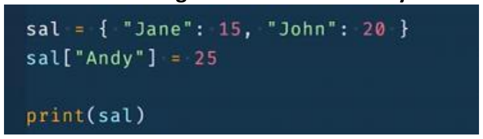
For loop in python



```
nums = [1, 2, 3, 4]
print(nums)

nums.append(5)
print(nums)

x = nums.pop(0)
print(x)
print(nums)

for x in nu
    print(x)
```

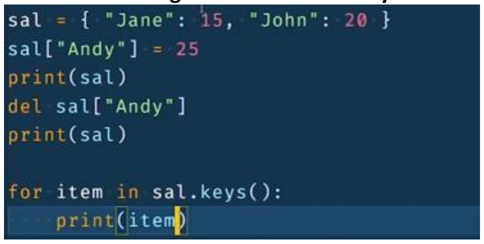
Then it will read the values in the array

To add some thing new in the dictionary



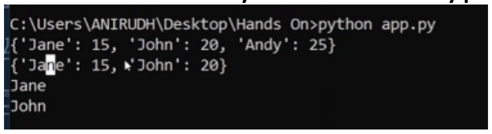
```
sal = { "Jane": 15, "John": 20 }
sal["Andy"] = 25

print(sal)
```

Now we are using for to the dictionary



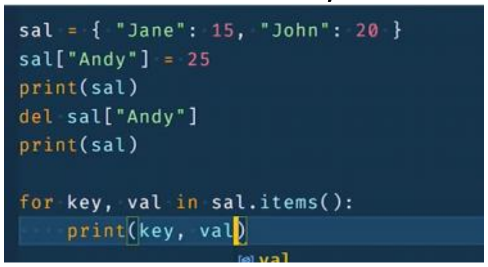
```
sal = { "Jane": 15, "John": 20 }
sal["Andy"] = 25
print(sal)
del sal["Andy"]
print(sal)

for item in sal.keys():
    print(item)
```

So now for will read only the value not he key pair of it like it will read john and jane

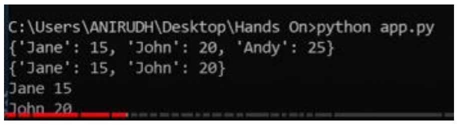


```
C:\Users\ANIRUDH\Desktop\Hands On>python app.py
{'Jane': 15, 'John': 20, 'Andy': 25}
{'Jane': 15, 'John': 20}
Jane
John
```

But in some we also need the key with it

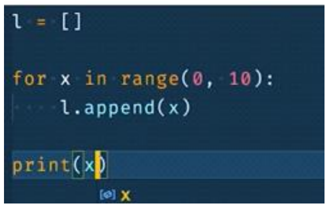


```
sal = { "Jane": 15, "John": 20 }
sal["Andy"] = 25
print(sal)
del sal["Andy"]
print(sal)

for key, val in sal.items():
    print(key, val)
```



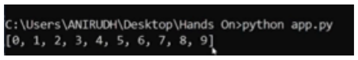
```
C:\Users\ANIRUDH\Desktop\Hands On>python app.py
{'Jane': 15, 'John': 20, 'Andy': 25}
{'Jane': 15, 'John': 20}
Jane 15
John 20
```

Here I am writing a program which will save automatically numbers in the empty list



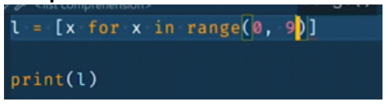
```
l = []

for x in range(0, 10):
    l.append(x)

print(x)
```

We need to print l not x



```
C:\Users\ANIRUDH\Desktop\Hands On>python app.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Compressed version of above code



```
l = [x for x in range(0, 9)]

print(l)
```