Datatypes

String

String Variable

```
In [100]:
1 x="What is your name?"
2 print(x)
```

```
what is your name?
```

```
In [101]:
1 x='What is your name?'
2 print(x)
```

```
What is your name?
```

```
In [135]:
1 x="What is your name?'
2 print(x)
```

```
  File "<ipython-input-135-11842970c32e>", line 1
    x="What is your name?'
                         ^
SyntaxError: EOL while scanning string literal
```

String Variable ...

In [132]:
```
1 x="""\tDear all,
2 Please find the attached file"""
3 print(x)
```

```
        Dear all,
Please find the attached file
```

In [133]:
```
1 x='''\tDear all,
2 Please find the attached file'''
3 print(x)
```

```
        Dear all,
Please find the attached file
```

String Variable ...

In [159]:
```
1 x="put several strings into parenthesis
2 to have them joined together"
```

```
  File "<ipython-input-159-b67d9294bb92>", line 1
    x="put several strings into parenthesis
                                           ^
SyntaxError: EOL while scanning string literal
```

In [167]:
```
1 x="put several strings into parenthesis "\
2 "to have them joined together"
3 print(x)
```

```
put several strings into parenthesis to have them joined together
```

String Variable...

```
[27]: print('"Isn\'t," they said.')
```

```
"Isn't," they said.
```

\n: means new line

```
[28]: print('First line.\nSecond line.')    # \n means newline
```

```
First line.
Second line.
```

If you don't want characters prefaced by \ to be interpreted as special characters, you can use raw strings by adding an r before the first quote

```
[29]: print('C:\some\name') # here \n means newline!
```

```
C:\some
ame
```

```
[31]: print(r'C:\some\name') # note the r before the quote
```

```
C:\some\name
```

String Variable ...

In [103]:
```
1 x="What's your name?"
2 print(x)
```

```
What's your name?
```

In [113]:
```
1 x='What's your name?'
2 print(x)
```

```
  File "<ipython-input-113-f7896ae28912>", line 1
    x='What's your name?'
           ^
SyntaxError: invalid syntax
```

In [114]:
```
1 x='What\'s your name?'
2 print(x)
```

```
What's your name?
```

String Variable ...

In [122]:
```
1 x='Ahmed says "I am so tired. " '
2 print(x)
```

```
Ahmed says "I am so tired. "
```

In [123]:
```
1 x="Ahmed says "I am so tired. " "
2 print(x)
```

```
  File "<ipython-input-123-241b9ee57954>", line 1
    x="Ahmed says "I am so tired. " "
                  ^
SyntaxError: invalid syntax
```

In [124]:
```
1 x="Ahmed says \"I am so tired. \" "
2 print(x)
```

```
Ahmed says "I am so tired. "
```

Basic String Operations: '+'

concatenating strings using + operator

[137]:
```
firstName='Zeinab'
lastName='Ezz'
fullName=firstName + ' '+ lastName
print(fullName)
```

```
Zeinab Ezz
```

[139]:
```
welcomeMsg='Welcome '+ firstName
print(welcomeMsg)
```

```
Welcome Zeinab
```

Basic String Operations: '+' ...

In [138]:
```
1 x=50+" cent"
2 print(x)
```

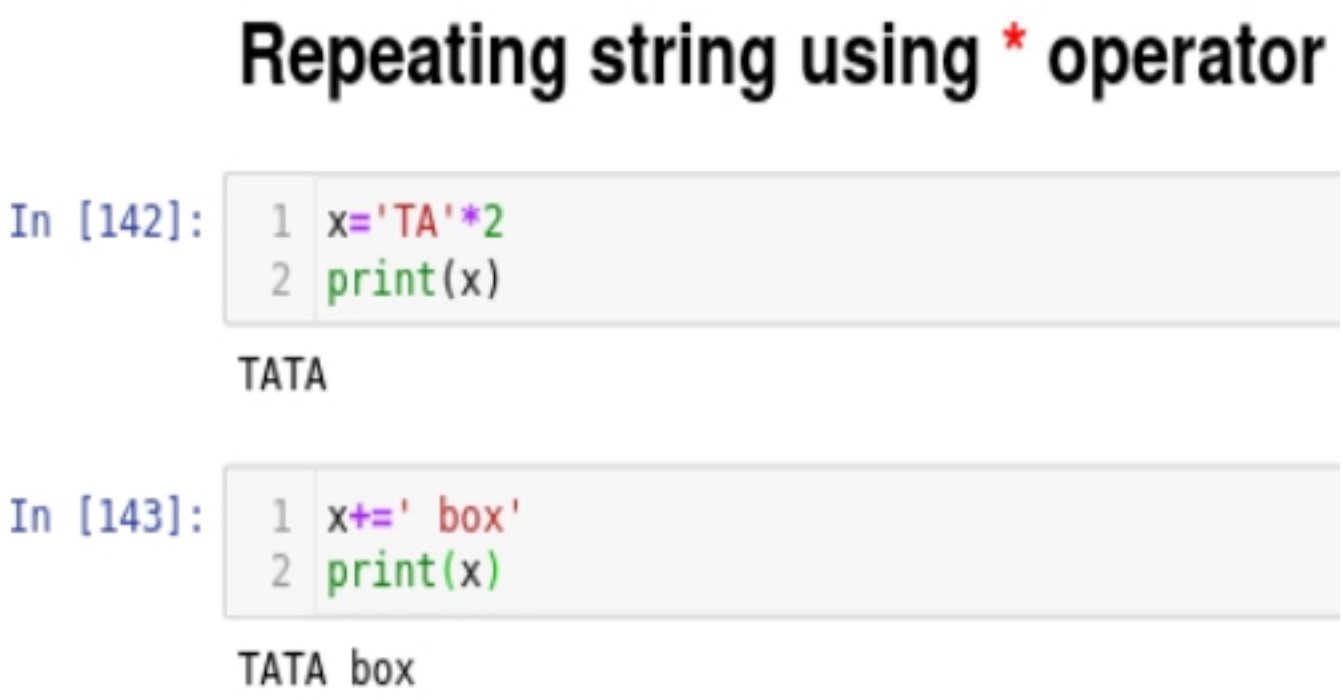
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-138-626ef9795668> in <module>
----> 1 x=50+" cent"
      2 print(x)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [139]:
```
1 x=str(50)+" cent"
2 print(x)
```

```
50 cent
```

Basic String Operations: ‘*’



in keyword

To check if a certain phrase or character is present in a string, we can use the keyword **in**.

```
In [144]:
1 x="hello world"
2 y='hello'
3 y in x

Out[144]: True
```

```
In [145]:
1 x="hello world"
2 y='Hello'
3 y in x

Out[145]: False
```

To check if a certain phrase or character is NOT present in a string, we can use the keyword **not in**

```
In [146]:
1 x="hello world"
2 y='hello'
3 y not in x

Out[146]: False
```

```
In [147]:
1 x="hello world"
2 y='Hello'
3 y not in x

Out[147]: True
```

String Comparison

1. >>> "january" == "jane"
   False
2. >>> "january" != "jane"
   True
3. >>> "january" < "jane"
   False
4. >>> "january" > "jane"
   True
5. >>> "january" <= "jane"
   False
6. >>> "january" >= "jane"
   True
7. >>> "filled" > ""
   True

Formating String: *fstring*

- A f-string is a string literal that is prefixed with “f”.
- These strings may contain replacement fields, which are expressions enclosed within curly braces {}.
- The expressions are replaced with their values.
- In the real world, it means that you need to specify the name of the variable inside the curly braces to display its value.
- An f at the beginning of the string tells Python to allow any currently valid variable names within the string.

```
In [1]: 1 country = input("Which country do you live in?")
        2 print(f"I live in {country}")

Which country do you live in?Egypt
I live in Egypt
```

Formating String: *format()*

Formating String using format() function

"string to be formatted".format(values or variables to be inserted into string, separated by commas)

[96]:
```
message = 'The price of this {0:s} laptop is {1:d} USD and the exchange rate is {2:4.2f} USD to 1 EUR'.format('Apple', 1299, 1.235235245)
print(message)
```

```
The price of this Apple laptop is 1299 USD and the exchange rate is 1.24 USD to 1 EUR
```

[100]:
```
message = 'The price of this {0} laptop is {1} USD and the exchange rate is {2:4.2f} USD to 1 EUR'.format('Apple', 1299, 1.235235245)
print(message)
```

```
The price of this Apple laptop is 1299 USD and the exchange rate is 1.24 USD to 1 EUR
```

[105]:
```
message = 'The price of this {} laptop is {} USD and the exchange rate is {:4.2f} USD to 1 EUR'.format('Apple', 1299, 1.235235245)
print(message)
```

```
The price of this Apple laptop is 1299 USD and the exchange rate is 1.24 USD to 1 EUR
```

[119]:
```
message = 'The price of this {2} laptop is {0} USD and the exchange rate is {1} USD to 1 EUR'.format('Apple', 1299, 1.235235245)
print(message)
```

```
The price of this 1.235235245 laptop is Apple USD and the exchange rate is 1299 USD to 1 EUR
```

[123]:
```
message = 'The price of this {:s} laptop is {:8d} USD and the exchange rate is {:6.2f} USD to 1 EUR'.format('Apple', 1299, 1.235235245)
print(message)
```

```
The price of this Apple laptop is     1299 USD and the exchange rate is   1.24 USD to 1 EUR
```

Formating String: *format( ) ...*

Formating String using format() function

"string to be formatted".format(values or variables to be inserted into string, separated by commas)

[59]:
```
age=25
txt="hello, I am {} years old."
print(txt.format(age))
```

```
hello, I am 25 years old.
```

[61]:
```
name='Ahmed'
CS=90
OR= 85
IT=70
txt="hello, My name is {}. My grade in CS is {}. My grade in IT is {} and in OR is {}."
print(txt.format(name, CS, IT, OR))
```

```
hello, My name is Ahmed. My grade in CS is 90. My grade in IT is 70 and in OR is 85.
```

[60]:
```
name='Ahmed'
CS=90
OR= 85
IT=70
txt="hello, My name is {0}. My grade in CS is {1}. My grade in IT is {3} and in OR is {2}."
print(txt.format(name, CS, OR, IT))
```

```
hello, My name is Ahmed. My grade in CS is 90. My grade in IT is 70 and in OR is 85.
```

String Functions

1- upper

```
[85]: upperFirstName=firstName.upper()
      print(upperFirstName)
      ZEINAB

[86]: "Python course".upper()
[86]: 'PYTHON COURSE'
```

2- lower

```
[83]: lowerFirstName=upperFirstName.lower()
      print(lowerFirstName)
      zeinab

[84]: "PYTHON COURSE".lower()
[84]: 'python course'
```

3 - strip()

Remove Whitespace. Whitespace is the space before and/or after the actual text.

```
[66]: a = "     Hello, World!     "
      print(a.strip()) # returns "Hello, World!"
      Hello, World!
```

4 - replace()

replaces a string with another string:

```
[67]: a = "Hello, World!"
      print(a.replace("H", "J"))
      Jello, World!
```

string functios: *join()*

string_name.join(sequence)

```
1. >>> date_of_birth = ["17", "09", "1950"]
2. >>> ":".join(date_of_birth)
   '17:09:1950'
3. >>> social_app = ["instagram", "is", "an", "photo", "sharing", "application"]
4. >>> " ".join(social_app)
   'instagram is an photo sharing application'
5. >>> numbers = "123"
6. >>> characters = "amy"
7. >>> password = numbers.join(characters)
8. >>> password
   'a123m123y'
```

String Functions: *split()*

The split() method returns a list of string items by breaking up the string using the delimiter string. The syntax of split() method is,

```
string_name.split([separator [, maxsplit]])
```

```
1. >>> inventors = "edison, tesla, marconi, newton"
2. >>> inventors.split(",")
   ['edison', ' tesla', ' marconi', ' newton']
3. >>> watches = "rolex hublot cartier omega"
4. >>> watches.split()
   ['rolex', 'hublot', 'cartier', 'omega']
```

Escape characters

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab

```
[72]: print( "We are the so-called "Vikings" from the north." )
```

```
  File "<ipython-input-72-18fc3f7f18b3>", line 1
    print( "We are the so-called "Vikings" from the north." )
                                  ^
SyntaxError: invalid syntax
```

```
[71]: print( "We are the so-called \"Vikings\" from the north." )
```

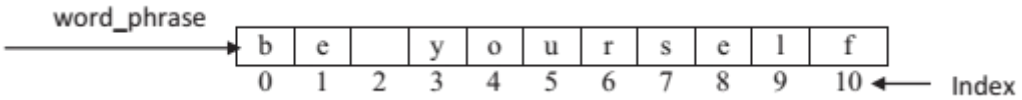
```
We are the so-called "Vikings" from the north.
```

String Positive Indexing

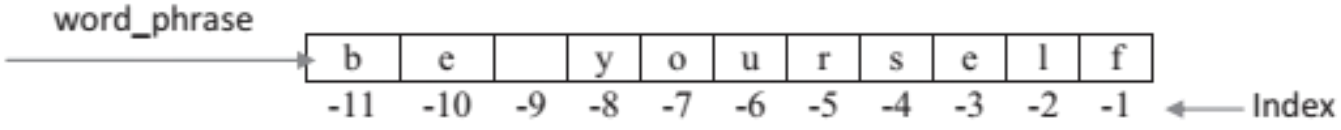
The syntax for accessing an individual character in a string is as shown below.

string_name[index]

```
1. >>> word_phrase = "be yourself"
2. >>> word_phrase[0]
   'b'
3. >>> word_phrase[1]
   'e'
4. >>> word_phrase[2]
   ' '
5. >>> word_phrase[3]
   'y'
6. >>> word_phrase[10]
   'f'
7. >>> word_phrase[11]
   Traceback (most recent call last):
     File "<stdin>", line 1, in <module>
   IndexError: string index out of range
```



String Negative Indexing



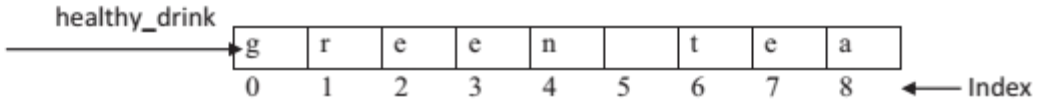
1. >>> word_phrase[-1]
   'f'
2. >>> word_phrase[-2]
   'l'

String Slicing

While indexing is used to obtain individual characters, slicing allows you to obtain substring.

The syntax for string slicing is, ***string_name[start:end[:step]]***

Specify the start index and the end index, separated by a colon, to return a part of the string.



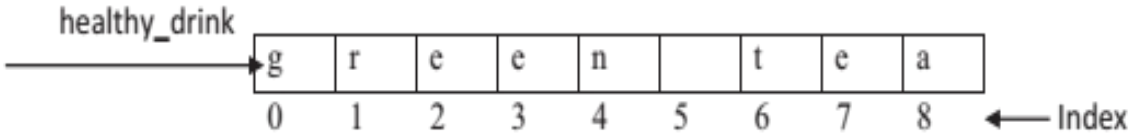

```
1. >>> healthy_drink = "green tea"
2. >>> healthy_drink[0:3]
   'gre'
```

String Slicing ...

Slice indices have useful defaults;

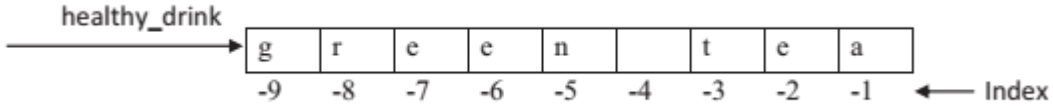
an omitted first index defaults to zero,

an omitted second index defaults to the size of the string being sliced.



3. >>> healthy_drink[:5]
   'green'
4. >>> healthy_drink[6:]
   'tea'
5. >>> healthy_drink[:]
   'green tea'
6. >>> healthy_drink[4:4]
   ''
7. >>> healthy_drink[6:20]
   'tea'

String Slicing ...



1. >>> healthy_drink[-3:-1]
   'te'
2. >>> healthy_drink[6:-1]
   'te'

Specifying Steps in Slice Operation

```
1. >>> newspaper = "new york times"
2. >>> newspaper[0:12:4]
   'ny'
3. >>> newspaper[::4]
   'ny e'
```

Strings Are Immutable

```
1. >>> immutable = "dollar"
2. >>> immutable[0] = "c"
   Traceback (most recent call last):
     File "<stdin>", line 1, in <module>
   TypeError: 'str' object does not support item assignment
3. >>> string_immutable = "c" + immutable[1:]
4. >>> string_immutable
   'collar'
5. >>> immutable = "rollar"
6. >>> immutable
   'rollar'
```

String built-in functions

Function	Description
capitalize()	Converts first character to Capital Letter
casefold()	converts to case folded strings
center()	Pads string with specified character
count()	returns occurrences of substring in string
encode()	returns encoded string of given string
endswith()	Checks if String Ends with the Specified Suffix
expandtabs()	Replaces Tab character With Spaces
find()	Returns the index of first occurrence of substring
format()	formats string into nicer output
format_map()	Formats the String Using Dictionary
index()	Returns Index of Substring
isalnum()	Checks Alphanumeric Character
isalpha()	Checks if All Characters are Alphabets
isdecimal()	Checks Decimal Characters
isdigit()	Checks Digit Characters

String built-in functions...

Function	Description
isnumeric()	Checks Numeric Characters
isprintable()	Checks Printable Character
isspace()	Checks Whitespace Characters
istitle()	Checks for Titlecased String
isupper()	returns if all characters are uppercase characters
join()	Returns a Concatenated String
ljust()	returns left-justified string of given width
lower()	returns lowercased string
lstrip()	Removes Leading Characters
maketrans()	returns a translation table
partition()	Returns a Tuple
replace()	Replaces Substring Inside
rfind()	Returns the Highest Index of Substring
rindex()	Returns Highest Index of Substring
rjust()	returns right-justified string of given width

String built-in functions...

Function	Description
rstrip()	Removes Trailing Characters
split()	Splits String from Left
splitlines()	Splits String at Line Boundaries
startswith()	Checks if String Starts with the Specified String
strip()	Removes Both Leading and Trailing Characters
swapcase()	swap uppercase characters to lowercase; vice versa
title()	Returns a Title Cased String
translate()	returns mapped charactered string
upper()	returns uppercased string
zfill()	Returns a Copy of The String Padded With Zeros
rpartition()	Returns a Tuple
rsplit()	Splits String From Right
isidentifier()	Checks for Valid Identifier
islower()	Checks if all Alphabets in a String are Lowercase

Excercise

Write Python Code to Determine Whether the Given String Is a Palindrome or Not Using Slicing

```
1. def main():
2.     user_string = input("Enter string: ")
3.     if user_string == user_string[::-1]:
4.         print(f"User entered string is palindrome")
5.     else:
6.         print(f"User entered string is not a palindrome")
7. if __name__ == "__main__":
8.     main()
```

OUTPUT

Case 1:
Enter string: madam
User entered string is palindrome

Collections

Lists

Collections: List

List: is a collection which is ordered and changeable. Allows duplicate members.

Lists are used to **store multiple items** in a single variable.

Lists elements can be of the **same or of different types.**

listName = (initial values)

Define Lists

[451]:
```
lst=[23, 'hello', 56.45, 6.5]
```

Empty List

[452]:
```
l=[]
```

Printing Lists

[453]:
```
print(lst)
```
```
[23, 'hello', 56.45, 6.5]
```

[454]:
```
print(l)
```
```
[]
```

Lists: Access list elements

List indexing: list indices starts from zero to the length of the list -1

```
[455]: lst[0]    #the first element
```

```
[455]: 23
```

```
[456]: lst[1]   #the second element
```

```
[456]: 'hello'
```

```
[457]: lst[2]      #the third element
```

```
[457]: 56.45
```

```
[458]: lst[3]     # the last element since the list contain 4 elements
```

```
[458]: 6.5
```

```
[459]: lst[6]    #6: out of range index, since the length contains 4 elements
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-459-e38b678c4f2f> in <module>
----> 1 lst[6]    #6: out of range index, since the length contains 4 elements

IndexError: list index out of range
```

Lists: Negative indexing

Negative indexing starts from the last element in the list, its index is -1

```
[460]: lst[-1]     # the last element in the list is accessed also via index -1
[460]: 6.5
[461]: lst[-2]
[461]: 56.45
[462]: lst[-3]
[462]: 'hello'
[463]: lst[-4]       #the first element
[463]: 23
[464]: lst[-5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-464-f2fcf4218453> in <module>
----> 1 lst[-5]

IndexError: list index out of range
```

Lists: Slicing

List Slicing to access a sublist using [start:end-1] notation

```
[465]: lst[1:4]

[465]: ['hello', 56.45, 6.5]

[466]: x=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
       x[2:7]

[466]: [3, 4, 5, 6, 7]

[467]: y=x[:5]            # default start index is 0
       print(y)

       [1, 2, 3, 4, 5]

[468]: x[3:]              #default end index is the size of the list-1

[468]: [4, 5, 6, 7, 8, 9, 10]

[469]: x[:]             #the whole list

[469]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

[470]: x[::2]     # with step =2

[470]: [1, 3, 5, 7, 9]

[471]: x[1:6:3]          #starts from index 1 to index 6 with step= 3

[471]: [2, 5]
```

Lists: Modify list

listName[index of item to be modified] = new value

```
[472]: x[5]
```

```
[472]: 6
```

```
[473]: x[5]='hello'
       print(x[5])
```

```
hello
```

```
[474]: print(x)
```

```
[1, 2, 3, 4, 5, 'hello', 7, 8, 9, 10]
```

```
[475]: x[:4]
```

```
[475]: [1, 2, 3, 4]
```

```
[476]: x[:4]=[11, 23.5, 'programming', 0]
       print(x[:4])
```

```
[11, 23.5, 'programming', 0]
```

```
[477]: x
```

```
[477]: [11, 23.5, 'programming', 0, 5, 'hello', 7, 8, 9, 10]
```

Lists: Concatenation

To concatenate two or more lists, + operator is used

```
[75]: squares = [1, 4, 9, 16, 25]
      squares

[75]: [1, 4, 9, 16, 25]

[80]: squares + [36, 49, 64, 81, 100]

[80]: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Unpack a Collection

- If you have a collection of values in a list, tuple etc. Python allows you extract the values into variables. This is called unpacking.

Example

- Unpack a list:

```
[38]: fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```
apple
banana
cherry
```

Lists: Clear list

[86]:
```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
letters
```

[86]:
```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

[87]:
```
letters[2:5] = ['C', 'D', 'E']
letters
```

[87]:
```
['a', 'b', 'C', 'D', 'E', 'f', 'g']
```

[88]:
```
# now remove them
letters[2:5] = []
letters
```

[88]:
```
['a', 'b', 'f', 'g']
```

[89]:
```
# clear the list by replacing all the elements with an empty list
letters[:] = []
letters
```

[89]:
```
[]
```

List Functions

List Length

```
[478]: len(x)
```

```
[478]: 10
```

```
[479]: len(lst)
```

```
[479]: 4
```

Adding item to List using append() function

```
[480]: x.append(234)    # add 234 to the end of list x
       print(x)
```

```
[11, 23.5, 'programming', 0, 5, 'hello', 7, 8, 9, 10, 234]
```

Check Existance

To check whether an element exist in a list or not, in keyword is used

```
[481]: print(x)
```

```
[11, 23.5, 'programming', 0, 5, 'hello', 7, 8, 9, 10, 234]
```

```
[485]: print (11 in x)
```

```
True
```

```
[486]: print(11.5 in x)
```

```
False
```

List Functions...

Removing item from List

del listName[index of item to be deleted]

[487]:
```
del x[2]
print(x)
```

```
[11, 23.5, 0, 5, 'hello', 7, 8, 9, 10, 234]
```

Removing List

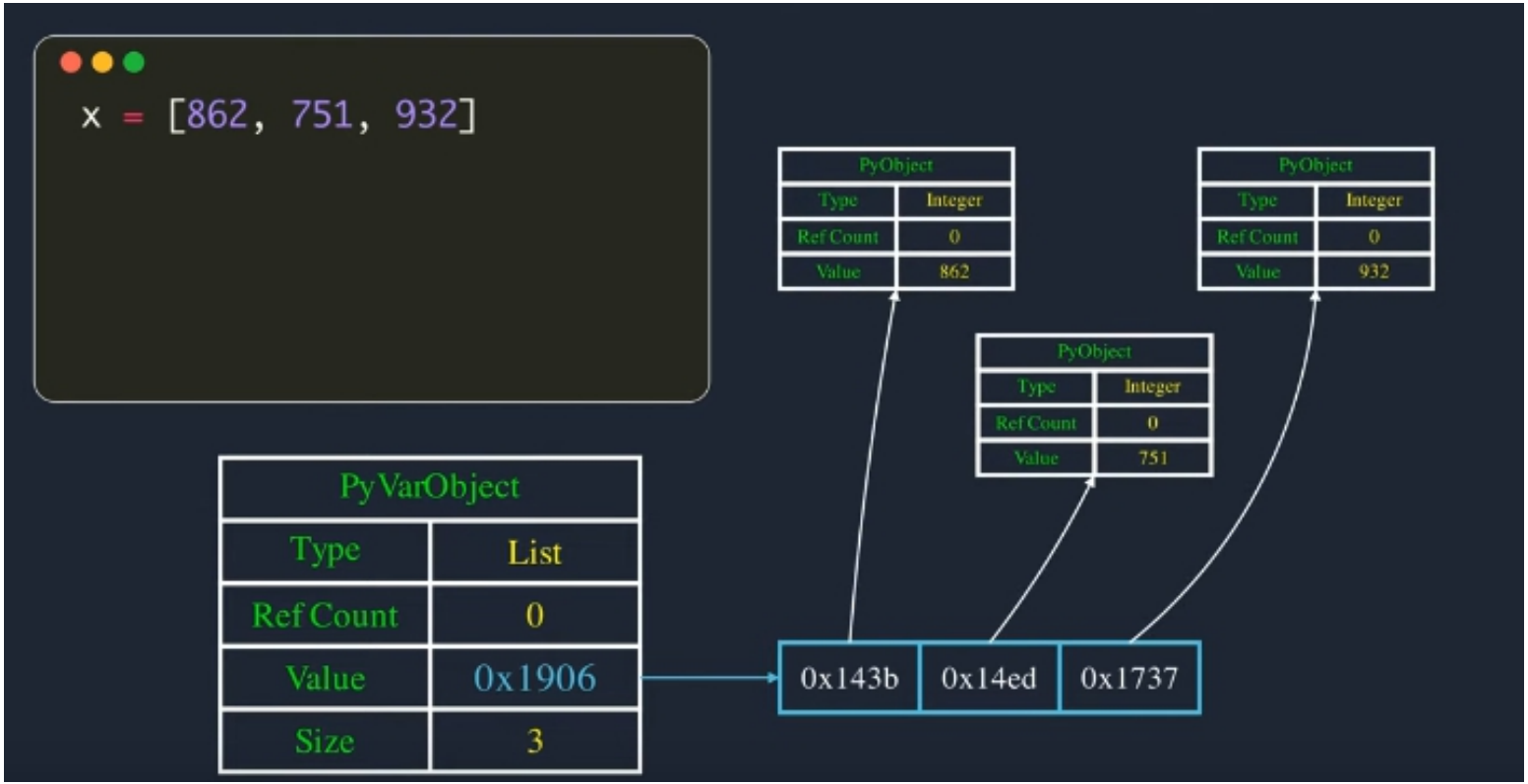
del listName

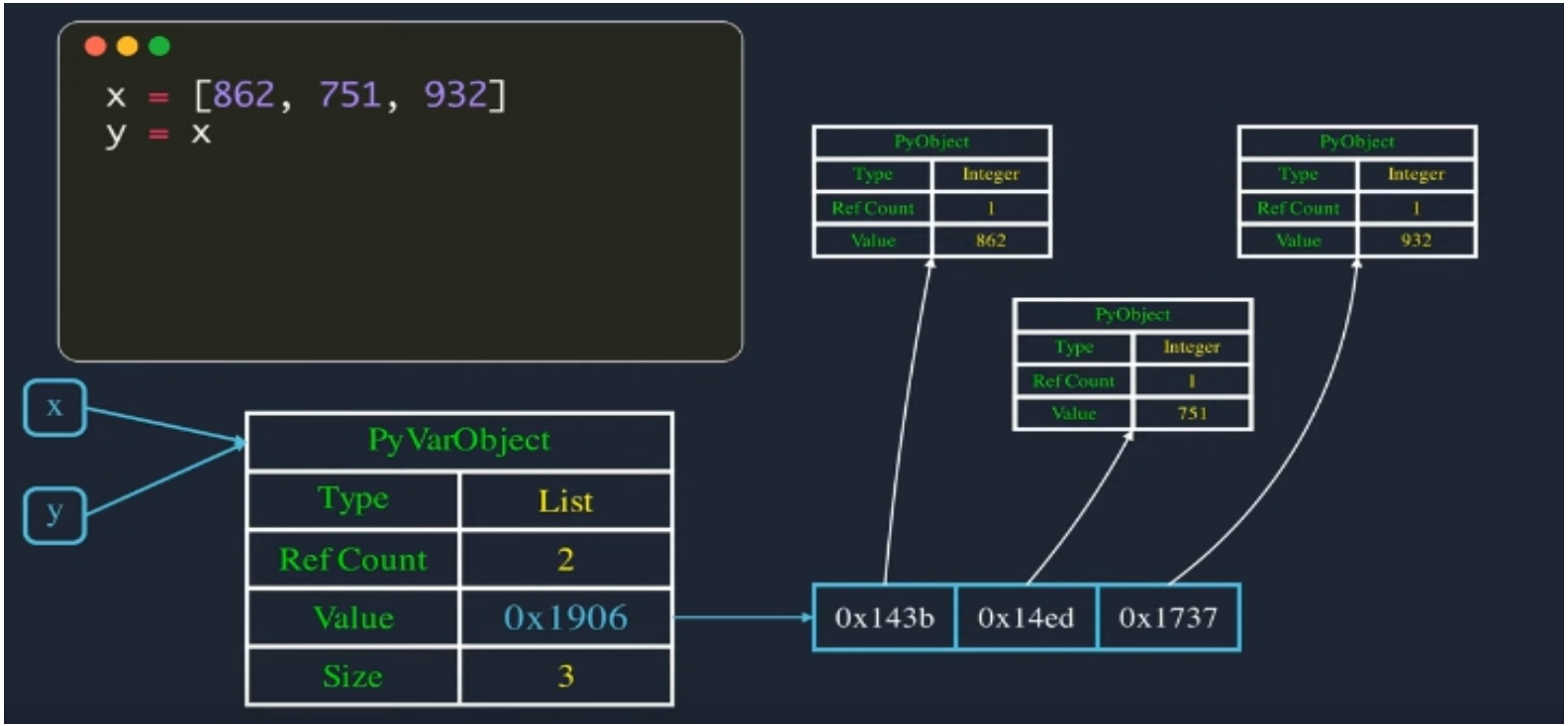
[488]:
```
del x
print(x)
```

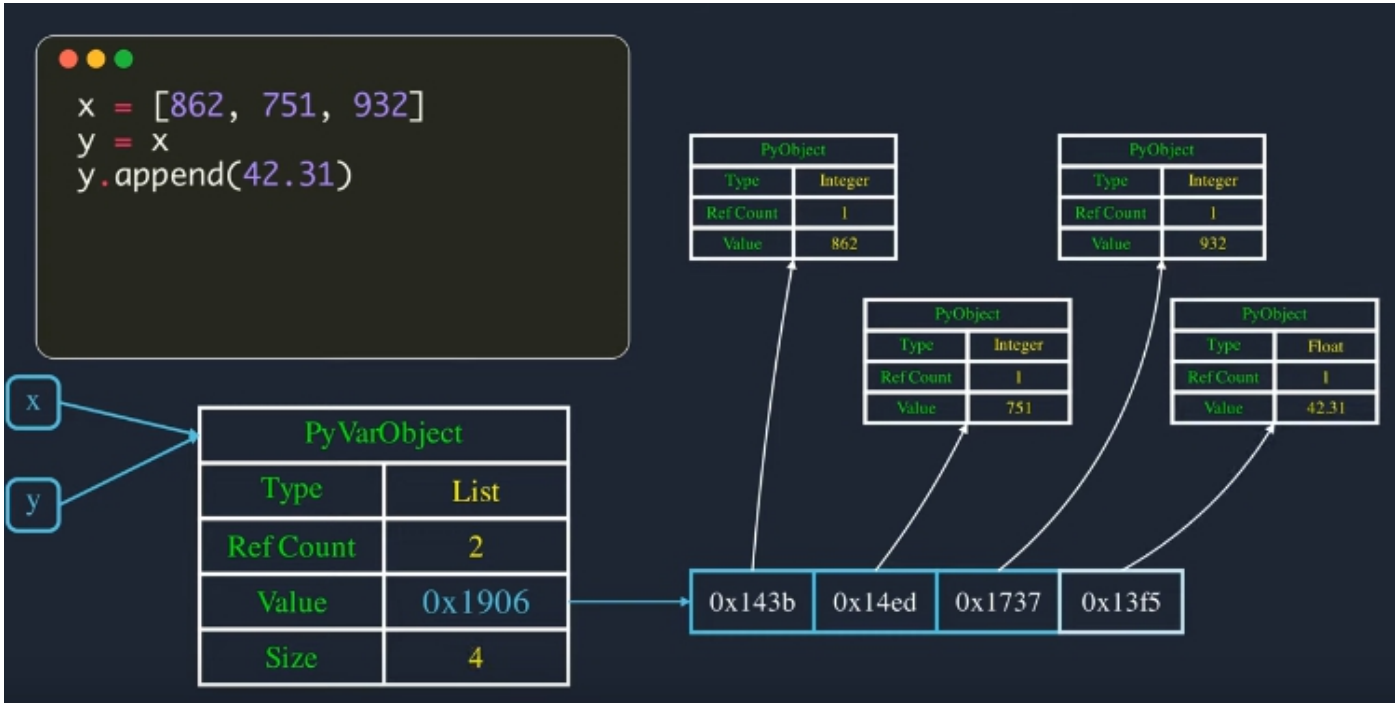
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-488-8c23cd95a3fd> in <module>
      1 del x
----> 2 print(x)

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

Storing Lists


x = [862, 751, 932]
PyObject
Type
Integer
Ref Count
0
Value
862
PyObject
Type
Integer
Ref Count
0
Value
932
PyObject
Type
Integer
Ref Count
0
Value
751
PyVarObject
Type
List
Ref Count
0
Value
0x1906
Size
3
0x143b
0x14ed
0x1737


x = [862, 751, 932]
y = x
x
y
PyVarObject
Type
List
Ref Count
2
Value
0x1906
Size
3
0x143b
0x14ed
0x1737
PyObject
Type
Integer
Ref Count
1
Value
862
PyObject
Type
Integer
Ref Count
1
Value
751
PyObject
Type
Integer
Ref Count
1
Value
932


x = [862, 751, 932]
y = x
y.append(42.31)
x
y
PyVarObject
Type
List
Ref Count
2
Value
0x1906
Size
4
0x143b
0x14ed
0x1737
0x13f5
PyObject
Type
Integer
Ref Count
1
Value
862
PyObject
Type
Integer
Ref Count
1
Value
751
PyObject
Type
Integer
Ref Count
1
Value
932
PyObject
Type
Float
Ref Count
1
Value
42.31

List built-in functions

Function	Description
append()	Add a single element to the end of the list
clear()	Removes all Items from the List
copy()	returns a shallow copy of the list
count()	returns count of the element in the list
extend()	adds iterable elements to the end of the list
index()	returns the index of the element in the list
insert()	insert an element to the list
pop()	Removes element at the given index
remove()	Removes item from the list
reverse()	reverses the list
sort()	sorts elements of a list

Tuple

Tuple

Tuple: is a collection which is ordered and unchangeable. Allows duplicate members.

tupleName = (initial values)

Define Tuple

[418]:
```
monthes=('Jan',1,  'Feb', 2, 'Mar', 3, 'Apr', 4, 'May', 5, 'Jun', 6, 'Jul', 7, 'Aug', 8, 'Sep', 9, 'Oct', 10, 'Nov', 11, 'Dec', 12)
```

Empty Tuple

[419]:
```
t=()
```

Printing Tuples

[420]:
```
print(monthes)
```

```
('Jan', 1, 'Feb', 2, 'Mar', 3, 'Apr', 4, 'May', 5, 'Jun', 6, 'Jul', 7, 'Aug', 8, 'Sep', 9, 'Oct', 10, 'Nov', 11, 'Dec', 12)
```

[421]:
```
print(t)
```

```
()
```

Tuple ...

Create a nested Tuple

```
[50]: # Tuples may be nested:
      u = t, (1, 2, 3, 4, 5)
      print(u)

      ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

Empty tuples

```
[60]: empty = ()
      len(empty)

[60]: 0
```

Tuple with one item is constructed by following a value with a comma

```
[61]: singleton = 'hello',     # <-- note trailing comma
      len(singleton)
      singleton

[61]: ('hello',)
```

Tuple: Access tuple elements

Tuple indexing: Tuple indices starts from zero to the length of the list -1

```
[422]: monthes[0]    #the first element
```

```
[422]: 'Jan'
```

```
[423]: monthes[1]   #the second element
```

```
[423]: 1
```

```
[424]: monthes[2]      #the third element
```

```
[424]: 'Feb'
```

```
[425]: monthes[23]     # the last element since the tuple contain 24 elements
```

```
[425]: 12
```

```
[426]: monthes[24]    #6: out of range index, since the tuple contains 24 elements
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-426-2b9052b31fcb> in <module>
----> 1 monthes[24]    #6: out of range index, since the tuple contains 24 elements

IndexError: tuple index out of range
```

Tuple: Negative indexing

Negative indexing starts from the last element in the tuple, its index is -1

```
[427]: monthes[-1]     # the last element in the tuple is accessed also via index -1
```

```
[427]: 12
```

```
[428]: monthes[-2]       # the second last element on the tuple
```

```
[428]: 'Dec'
```

```
[429]: monthes[-3]
```

```
[429]: 11
```

```
[430]: monthes[-24]       #the first element
```

```
[430]: 'Jan'
```

```
[431]: monthes[-25]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-431-884588bd498d> in <module>
----> 1 monthes[-25]

IndexError: tuple index out of range
```

Tuple: Slicing

Tuple Slicing to access a subtuple using [start:end-1] notation

```
[432]: monthes[1:4]

[432]: (1, 'Feb', 2)

[433]: days=('sat', 'sun', 'mon', 'tues', 'wed', 'thrus', 'fri')
       days[2:7]

[433]: ('mon', 'tues', 'wed', 'thrus', 'fri')

[434]: y=days[:5]            # default start index is 0
       print(y)

       ('sat', 'sun', 'mon', 'tues', 'wed')

[435]: days[3:]              #default end index is the size of the list-1

[435]: ('tues', 'wed', 'thrus', 'fri')

[436]: days[:]             #the whole list

[436]: ('sat', 'sun', 'mon', 'tues', 'wed', 'thrus', 'fri')

[437]: days[::2]     # with step =2

[437]: ('sat', 'mon', 'wed', 'fri')

[438]: days[1:6:3]          #starts from index 1 to index 6 with step= 3

[438]: ('sun', 'wed')
```

Tuple:modify tuples

Tuple elements cannot be modified after created

```
[439]: monthes[5]

[439]: 3

[440]: monthes[5]='hello'
       print(monthes[5])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-440-134bed95031b> in <module>
----> 1 monthes[5]='hello'
      2 print(monthes[5])

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

```
[441]: monthes[:4]

[441]: ('Jan', 1, 'Feb', 2)

[442]: monthes[:4]=(11, 23.5, 'programming', 0)
       print(monthes[:4])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-442-a5e109474cc4> in <module>
----> 1 monthes[:4]=(11, 23.5, 'programming', 0)
      2 print(monthes[:4])

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

Tuple: Concatenation

To concatenate two or more tuple, + operator is used

```
[30]: t1= (3, 5, 2)
      t2= 4,
      t3= 9, 1, 7, 5

      t=t1+t2+t3
      print(t)
```

```
(3, 5, 2, 4, 9, 1, 7, 5)
```

Tuple Functions

Tuple Length

```
[443]: len(monthes)
```

```
[443]: 24
```

```
[444]: len(t)
```

```
[444]: 0
```

Adding item to Tuple

We cannot add a new item to a tuple after created

```
[445]: monthes[24]=67    # add 67 to the end of tuple monthes
       print(monthes)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-445-2d50ca17b5c9> in <module>
----> 1 monthes[24]=67    # add 67 to the end of tuple monthes
      2 print(monthes)

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

Tuple Functions ...

Check Existance

To check whether an element exist in a tuple or not, in keyword is used

[446]:
```
print(days)
```
```
('sat', 'sun', 'mon', 'tues', 'wed', 'thrus', 'fri')
```

[448]:
```
print ('sat' in days)
```
```
True
```

[450]:
```
print('moon' in days)
```
```
False
```

Tuple Functions ...

Removing item from Tuple

Cannot delete any elemant from the tuple

[256]:
```
del monthes[2]
print(monthes)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-256-30aa2068b6b2> in <module>
----> 1 del monthes[2]
      2 print(monthes)

TypeError: 'tuple' object doesn't support item deletion
```

Removing Tuple

del tupleName

[257]:
```
del monthes
print(monthes)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-257-432f39ba9814> in <module>
      1 del monthes
----> 2 print(monthes)

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

Tuple built-in functions

Function	Description
count()	returns count of the element in the tuple
index()	returns the index of the element in the tuple

Sets

Sets

Set: is a collection which is unordered and unindexed. No duplicate members.

setName={items}

Define Set

[327]:
```
numbers={'one', 'two', 2, 2, 'three'}
fruits=set({'orange', 'bananna', 'apple', 'cherry'})
```

Empty Sets

[328]:
```
s=set()                    # Note: donot use {} to create empty set
```

Printing Sets

[329]:
```
print(numbers)
```

```
{'three', 'one', 2, 'two'}
```

[330]:
```
print(fruits)
```

```
{'bananna', 'cherry', 'orange', 'apple'}
```

[331]:
```
print(s)
```

```
set()
```

Set Operations

```
[74]: a= set('abracadabra')    # unique letters in a
      a

[74]: {'a', 'b', 'c', 'd', 'r'}

[75]: b= set('alacazam')
      b

[75]: {'a', 'c', 'l', 'm', 'z'}

[76]: a-b      # letters in a but not in b

[76]: {'b', 'd', 'r'}

[77]: a|b     # letters in a or b or both

[77]: {'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}

[78]: a&b      # letters in both a and b

[78]: {'a', 'c'}

[79]: a^b      # letters in a or b but not both

[79]: {'b', 'd', 'l', 'm', 'r', 'z'}
```

```
[81]: 'orange' in basket
      # fast membership testing

[81]: True

[82]: 'crabgrass' in basket

[82]: False
```

Set: Access set elements

We cannot access a certain element by index in a set. This is because sets are unordered thus its element has no index

```
[332]: fruits[0]    #the first element
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-332-e15cc7d51d47> in <module>
----> 1 fruits[0]    #the first element

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

Sets: Negative indexing

Negative indexing sets have no negative indexing

```
[333]: fruits[-1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-333-5159a0c9a3df> in <module>
----> 1 fruits[-1]

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

Sets: Slicing

Set Slicing Sets donot support slicing

```
[334]: fruits[1:4]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-334-ddc1f3538d01> in <module>
----> 1 fruits[1:4]

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

Sets: modifying sets

Set Modification

Sets elements cannot be modified after created

```
[335]: fruits[2]='pineapple'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-335-5115c302126b> in <module>
----> 1 fruits[2]='pineapple'

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

```
[336]: fruits[2:4]={11, 23}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-336-ccfeb22656ea> in <module>
----> 1 fruits[2:4]={11, 23}

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

Sets: Concatenation

To concatenate two or more sets in a new set, union( ) function is used

```
[47]: s1={3, 5, 'hi', 89}
      s2={89, 56, 5, 'hi'}
      s3=s1.union(s2)

[48]: print(s3)

      {3, 5, 'hi', 56, 89}
```

Sets: Set functions

Sets Length

[337]:
```
len(fruits)
```

[337]:
```
4
```

[338]:
```
len(s)
```

[338]:
```
0
```

Adding item to Set using add() function

[339]:
```
fruits.add('pear')    # add pear to the set fruits
print(fruits)
```

```
{'bananna', 'apple', 'cherry', 'orange', 'pear'}
```

Adding multiple items to a set using update() function

[340]:
```
fruits.update({'mango', 'Grape'})
print(fruits)
```

```
{'bananna', 'apple', 'cherry', 'mango', 'Grape', 'orange', 'pear'}
```

Sets: Set functions ...

Check Existance

To check whether an element exist in a set or not, in keyword is used

```
[274]: print(fruits)
{'bananna', 'cherry', 'orange', 'apple'}

[275]: print ('orange' in fruits)
True

[277]: print('pineapple' in fruits)
False
```

discard()

remove an item from a set.

```
[2]: thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
{'apple', 'cherry'}

[3]: thisset.discard("pineapple")
```

Sets: Set functions ...

Emtpies a Set using clear() function

[350]:
```
fruits.clear()
print(fruits)
```

```
set()
```

Removing set

del setName

[351]:
```
del fruits
print(fruits)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-351-b9815f27b187> in <module>
      1 del fruits
----> 2 print(fruits)

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

Dictionary

Dictionaries

- Dictionaries are indexed by keys.
- keys can be any immutable type:
  - strings
  - numbers
  - Tuples can be used as keys if they contain only strings, numbers, or tuples
- if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.
- You can't use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().

Dictionary

Define Dictionary

[397]:
```
tel={'jack':12345, 'Alex':67890}    #jack is a key with value 12345, ALex is a key with value 67890
myDict={'Ali':38, "Maha":51, 'Ali':13} # the key Ali had the value 38, then it is replaced with valued 13
grade={500:60, 501:70, 502:80}     #key can be numbers
x={600:36, 'a':23, 700:'e'}      #both key and value can be number or string in the same dictionary
```

Empty Dictionary

[398]:
```
d={}
```

Printing Dictionary

[399]:
```
print(myDict)
```
```
{'Ali': 13, 'Maha': 51}
```

[400]:
```
print(d)
```
```
{}
```

[401]:
```
print(x)
```
```
{600: 36, 'a': 23, 700: 'e'}
```

[402]:
```
print(grade)
```
```
{500: 60, 501: 70, 502: 80}
```

Dictionary: Access dictionary elements

Dictionary indexing: Dictionary indices are its keys

```
[403]: myDict['Maha']    #return the value of the key 'Maha'

[403]: 51

[404]: x[700]   #return the value of the key 700

[404]: 'e'

[405]: x[2]      #ERROR!! since there is no element with key=2 in dictionary x

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-405-303fa26708b3> in <module>
----> 1 x[2]      #ERROR!! since there is no element with key=2 in dictionary x

KeyError: 2
```

Dictionary: Add item to dictionary

dictionaryName[dictionary key] = data

[406]:
```
tel['Mona']=87213    # add a new key value pair
print(tel)
```

```
{'jack': 12345, 'Alex': 67890, 'Mona': 87213}
```

Dictionary: Modifying dictionary

dictionaryName[dictionary key of item to be modified] = new data.

[407]:
```
x[2]=155      #create new key (2) with value (155) in dictionary (x)
print(x)
```

```
{600: 36, 'a': 23, 700: 'e', 2: 155}
```

[408]:
```
x[600]='hello'        #replace the value of key 600 with 'hello'
print(x)
```

```
{600: 'hello', 'a': 23, 700: 'e', 2: 155}
```

[409]:
```
myDict['Ali']=26
print(myDict)
```

```
{'Ali': 26, 'Maha': 51}
```

Dictionary: functions

Dictionary Length

```
[410]: len(x)
```

```
[410]: 4
```

```
[411]: len(myDict)
```

```
[411]: 2
```

Check Existance

To check whether an element exist in a dictionary or not, in keyword is used

```
[414]: print(tel)
```

```
{'jack': 12345, 'Alex': 67890, 'Mona': 87213}
```

```
[415]: print ('jack' in tel)
```

```
True
```

```
[417]: print('Martin' in tel)
```

```
False
```

Performing ***list(d)*** on a dictionary returns a list of all the keys used in the dictionary,

```
[95]: d=list(tel)
```

```
['jack', 'guido']
```

Dictionary: functions ...

Removing item from Dictionary

del dictionaryName[dictionary key]

[412]:
```
del x[2]
print(x)
```

```
{600: 'hello', 'a': 23, 700: 'e'}
```

Removing Dictionary

del dictionaryName

[396]:
```
del x
print(x)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-396-8c23cd95a3fd> in <module>
      1 del x
----> 2 print(x)

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

Dictionary built-in functions

Function	Description
clear()	Removes all Items
copy()	Returns Shallow Copy of a Dictionary
fromkeys()	creates dictionary from given sequence
get()	Returns Value of The Key
items()	returns view of dictionary's (key, value) pair
keys()	Returns View Object of All Keys
pop()	removes and returns element having given key
popitem()	Returns & Removes Latest Element From Dictionary
setdefault()	Inserts Key With a Value if Key is not Present
update()	Updates the Dictionary
values()	returns view of all values in dictionary