Data types in python

- numbers
- variables
- strings
- arrays
- list
- dictionary
- set

Types of Numbers

- Integers
- Floating-Point numbers
- Complex numbers

Integers

- Integers are the number such as 1, 8, -6, 100,...
- Integer arithmetic is exact

Floating-Point Numbers

- These are the representation of real numbers
- In general, the representation do not have the exact value is approximated value
- Any single number containing a period (".") is considered by Python to specify a floating-point number

```
4/3
```

```
1.3333333333333333
```

```
6-5.7
```

```
0.2999999999999998
```

Complex Numbers

- Numbers of the form $a + bj$
- It consists of real part and imaginary part
- Each part of complex number is a floating-point number
- Complex number arithmetic is not exact but subject to the same finite precision considerations as floats.

```
complex(2,6)
```

```
(2+6j)
```

```
a=2+6j
type(a)
```

```
complex
```

Basic Arithmetic

- Addition +
- Subtraction -
- Multiplication *
- Floating-point division /
- Integer Division (quotient) //
- Modulus (remainder) %
- Exponentiation

```
# Addition
2+3
```

```
5
```

```
# Subtraction
2-3
```

```
-1
```

```
# Multiplication
2*3
```

```
6
```

```
# Floting-point division
2/3
```

```
0.6666666666666666
```

```
# Integer division
# Always rounds down the result to
# the nearest smaller integer
# Also known as floor division
2//3
```

```
0
```

```
8.5//3
```

```
2.0
```

```
5//2
```

```
2
```

```
5%2
```

```
1
```

```
5**2
```

```
25
```

Methods and Attributes of Numbers

Python numbers are objects and have certain attributes, which are accessed using the "dot". For instance, complex number objects have the attributes real and imag, which are real and imaginery parts of the number.

```
a = (2+6j)
a.conjugate()
```

```
(2-6j)
```

```
type(a)
#dir(a)
```

```
complex
```

```
dir(a)
```

```
['__abs__',
 '__add__',
 '__bool__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__float__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
```

```
 '__int__',
 '__le__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
 '__pos__',
 '__pow__',
 '__radd__',
 '__rdivmod__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rfloordiv__',
 '__rmod__',
 '__rmul__',
 '__rpow__',
 '__rsub__',
 '__rtruediv__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__truediv__',
 'conjugate',
 'imag',
 'real']
```

```
(7+4j).imag
```

```
4.0
```

Other attributes of complex number objects are `methods`: callable functions that act on their object in some way. For instance, `conjugate` is a method for the object complex number.

```
(7+4j).conjugate()
```

```
(7-4j)
```

Here, the empty parentheses indicate that the method is to be `called`, i.e., the function to calculate the complex conjugate is to be run on the number $7 + 4j$

Mathematical Functions

Built-in mathe matical functions are `abs` and `round`.

```
abs(-5.7)
```

```
5.7
```

```
abs(3+5j)
```

```
5.830951894845301
```

```
abs(-2)
```

```
2
```

```
abs(3+4j)
```

```
5.0
```

```
round(-9.72)
```

```
-10
```

```
round(4.51)
```

```
5
```

```
round(4.567894,3)
```

```
4.568
```

Python is a very modular language: functionality is available in packages and modules that are `imported` if they are needed but are not loaded by default: this keeps the memory required to run a Python program to a minimum and improves performance.

For instance, many useful mathematical functions are provided by the `math` module, which is imported with the statement `import math`

```
import math
#import numpy
dir(math)
```

```
math.cos(0)
```

```
1.0
```

```
math.sqrt(16)
```

```
4.0
```

```
math.pi
```

```
3.141592653589793
```

```
math.factorial(5)
```

```
120
```

For more details about math module you can visit its documentation.

Variables

When an object created (like `float`) in a Python, memory is allocated for it: the location of this memory within the computer's architecture is called `address`

> A `variable` is a name that refers to a value.

A variable name can be assigned ("bound") to any object and used to identify that object in future calculations

```
id(5.3)
```

```
2533340724720
```

```
a = 5.3
print(id(5.3))
print(id(a))
```

```
2084757619920
2084757620528
```

This number refers to a specific location in memory that has been allocated to hold the float object with the value 5.3.

```
a = 3
b = -0.5
c = a*b
c
```

```
-1.5
```

Note that we did not have to declare the variables before we assign them (tell Python that the variable name `a` is to refer to an integer, `b` is to refer to a floating-point number, etc.), as is necessary in some computer languages.

Variable Names

There are some rules about what makes a valid variable name:

- Variable names are *case-sensitive*: a and A are different variables;
- Variable names can contain any letter, the underscore character ("_") and any digit (0–9)
- ... but must not *start with* a digit;
- A variable name must not be the same as one of the reserved keywords given in given Table.

- The built-in constant names `True`, `False` and `None` cannot be assigned as variable names.
- There are total 35 reserved key words in Python, you can't use as variable name.

```
and       del       from      None      True
as        elif      global    nonlocal  try
assert    else      if        not       while
break     except    import    or        with
class     False     in        pass      yield
continue  finally   is        raise     async
def       for       lambda    return    await
```

Good practice in naming variables:

- Variable names should be meaningful (area is better than a) but not too long (the_area_of_the_triangle is unwieldy);
- Generally, don't use I (upper-case i), l (lower-case L) or the upper-case letter O: they look too much like the digits 1 and 0;
- The variable names i, j and k are usually used as integer counters;
- Use lower-case names, with words separated by underscores rather than "CamelCase": for example, *mean_height* and not *MeanHeight*.

```
# Example
a = 4.5
b = 2
c = 3.902
d = 4.12
s = (a + b + c)/2
#area of the triangle
#Heron's formulae
area = math.sqrt(s * (s-a)*(s-b)*(s-c))
area
```

```
3.8935866886713875
```

```
type(s)
```

```
float
```

```
id(s)
```

```
2533356147184
```

```
id(area)
```

```
1904610635632
```

Operators

The main comparison operators used in Python to compare objects are given below:

Operator	
==	Equal to
!=	Not Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
7 == 8
```

```
False
```

```
4 >= 3.14
```

```
True
```

- The single equals sign is an *assignment*, which doesn't return a value
- The double equals sign is a logical statement which return `True` or `False`.

```
a = 0.01
b = 0.1**2
```

```
print(b)
a == b
```

```
0.010000000000000002
False
```

```
(0.1)**2
```

```
0.010000000000000002
```

```
c = 4.5
c = a
c
```

```
0.01
```

Logical Operators

Comparisons can be modified and strung together with the logic operator keywords `and`, `not` and `or`.

```
7.0 > 4 and -1 >= 0
```

```
False
```

```
5 < 4 or 1 != 2
```

```
True
```

```
not 7.5 < 0.9
```

```
True
```

```
not 7.5 < 0.9 or 4 == 4
```

```
True
```

```
not(7.5 < 0.9 or 4 == 4)
```

```
False
```

```
not(7.5 < 0.9 and 4 == 4)
```

```
True
```

Strings

- A Python string onject is an ordered, immutable sequence of characters.
- To define a variable containing some string, enclose the text in either single or double quotes.
- Strings used to store text data in python

```
greet = Sairam
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-a5259905d415> in <module>
----> 1 greet = Sairam

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

SEARCH STACK OVERFLOW

```
greet = "Sairam, I'm Sampath!"
greet
print(greet)
```

```
Sairam, I'm Sampath!
```

```
type(greet)
```

```
str
```

```
# + operator useful to perform adding strings
'Sairam'+ 'Sampath'
```

```
'SairamSampath'
```

```
'Sairam'+' '+  'Sampath'
```

```
'Sairam Sampath'
```

Python doesn't place any restriction on the length of a line. However, it's recommended that the maximum length of each line string of length upto 79 characters. To break up a string over two or more lines of code, use the line continuation character, "" or enclose the string literal in parentheses.

```
# Multiple lines
long_string = 'Sairam brothers, we continue to learning python,'\
'during the course, we also solve problems from MDSC-101,102 and 103...'
```

```
long_string
```

```
'Sairam brothers, we continue to learning python,during the course, we also solve problems from MDSC-101,102 and 103...'
```

```
# empty string
s = ''
s
```

```
''
```

```
# Converting int to string
str(42)
a ='42'
type(a)
```

```
str
```

```
int(a)
```

```
42
```

```
int(greet)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-33f627e595b3> in <module>
----> 1 int(greet)

ValueError: invalid literal for int() with base 10: "Sairam, I'm Sampath!"
```

SEARCH STACK OVERFLOW

```
str(3.4e-2)
```

```
'0.034'
```

```
str(3.4e2)
```

```
'340.0'
```

Escape Sequences

- Do you want include both quotes in a string?
- Do you want include more than one line in the string?

This case is handled by special escape sequences indicated by a backslash, .

Common Python escape sequences

Escape sequence	Meaning
'	Single quote(')
"	Double quote(")
\n	Linefeed(LF)
\r	Carriage return
\t	Horizontal tab
\b	Backspace

Escape sequence	Meaning
\	The backslach character itself
\u, \U, \N{}	Unicode character
\x	Hex-encoded byte

```
sentence = "He said, \bThis parrot's dead."
print(sentence)
```

```
He said,This parrot's dead.
```

```
print(sentence)
```

```
He said, "This parrot's dead."
```

```
subjects = 'MDSC-101\nMDSC-102\nMDSC-103\nMDSC-106'
subjects
```

```
'MDSC-101\nMDSC-102\nMDSC-103\nMDSC-106'
```

```
print(subjects)
```

```
MDSC-101
MDSC-102
MDSC-103
MDSC-106
```

Note:

- By typing a variable's name at the Python shell prompt simply echoes its literal value back to you (in quotes).
- To produce the desired string including the proper interpretation of special characters, pass the variable to the `print` built-in function.

If you want to define a string to include character sequences such as "\n" without them being escaped, define a raw string prefixed with `r`.

```
rawstring = r'The escape sequence for a new line \n.'
rawstring
```

```
'The escape sequence for a new line \\n.'
```

```
print(rawstring)
```

```
The escape sequence for a new line \n.
```

Triple-quoted strings

When defining a block of text including several line endings it is often inconvenient to use \n repeatedly. This can be avoided by using *triple-quoted* strings: new lines defined within strings delimited by """ and ''' are preserved in the string.

```
a = """
Hello!!
How are you?
Sairam.
"""
a
```

```
'\nHello!!\nHow are you?\nSairam.\n'
```

```
print(a)
```

```
Hello!!
How are you?
Sairam.
```

```
'\b'
```

```
'\x08'
```

Indexing and Slicing Strings

- *Indexing* a string returns a single character at a given location. Like all sequences in Python, strings are indexed with the first character having the index 0;

- *Slicing* a string, $s[i:j]$, produces a substring of a string between the characters at two indexes, including the first (i) but excluding the second (j);

```
a = 'Knight'
a[0]
```

```
'K'
```

```
a[-1]
```

```
't'
```

```
a[-4]
```

```
'i'
```

```
a[6]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-36-84c1e027215d> in <module>
----> 1 a[6]

IndexError: string index out of range
```

SEARCH STACK OVERFLOW

```
a[:3]
```

```
'Kni'
```

```
a[:3]
```

```
'Kni'
```

```
a[3:]
```

```
'ght'
```

```
a = "Studing python"
len(a)
```

```
14
```

```
a[1:11]
```

```
'tuding pyt'
```

```
a[::1]
```

```
'Studing python'
```

```
a[::-1]
```

```
'thginK'
```

```
a[:3]+a[3:] == a
```

```
True
```

```
a[3:10]
```

```
'ght'
```

```
# To test if a string contains a given substring
'kni' in a
```

```
False
```

```
'kni' in a
```

```
False
```

```
dir(a)
```

```
# Stride in a string
# To return every kth letter,
# set the stride to k.

s = 'Hemanth Sai Pavan'
s # striding 2 over s
```

```
'Hemanth Sai Pavan'
```

```
s.replace(s[3],'b')
```

```
'Hembnth Sbi Pbvbn'
```

```
s[::3]
```

```
'Re nas'
```

```
s[1::2]
```

```
'oetLnlns'
```

```
# Reversing a string
s[::-1]
```

```
'sdnalgnaL treboR'
```

String Methods

- Python strings are *immutable* objects.
- It's NOT possible to change a string by assignment
- New strings can be constructed from existing strings, but only as new objects
- To find the number of characters a string contains, we use `len` built-in method

```
a = 'Knight'
a[0] = 'b'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-176d5fae7402> in <module>
      1 a = 'Knight'
----> 2 a[0] = 'b'

TypeError: 'str' object does not support item assignment

SEARCH STACK OVERFLOW
```

```
a += ' King'
a
```

```
'Knight KingKing'
```

```
a
```

```
'Knight King'
```

```
len(a)
```

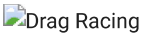
```
15
```

```
s= "HemanthSaiPavan"
```

```
s.join("Kumar")
```

```
'KHemanthSaiPavanuHemanthSaiPavanmHemanthSaiPavanaHemanthSaiPavanr'
```

Some common string Methods




```
dir(a)
```

```
a.lower()
```

```
'knight king king'
```

```
a.upper()
```

```
'KNIGHT KING KING'
```

```
a
```

```
'Knight King King'
```

```
a.replace('King','Queen')
```

```
'Knight Queen Queen'
```

```
a.split()
```

```
['Knight', 'King', 'King']
```

The `print` function

- In Python 3, `print` is a built-in function
- We discuss two standard string formating, number formatting methods

```
'{} plus {} equals {}'.format(2,3,'five')
```

```
'2 plus 3 equals five'
```

```
a =134
'a={0:5d}'.format(a) #decimal
```

```
'a=  134'
```

```
'a={0:10b}'.format(a) #binary
```

```
'a=  10000110'
```

```
'a={0:5f}'.format(a)
```

```
'a=134.000000'
```

```
print('{0:5d}\n{1:5d}\n={2:+3d}'.format(-4510,1001,-3026))
```

```
-4510
 1001
=-3026
```

```
a = 1.464e-7
'{0:g}'.format(a)
```

```
'1.464e-07'
```

```
a = 134
```

```
'{0:4f}'.format(a)
```

```
'134.000000'
```

```
'{0:10f}'.format(a)
```

```
'134.000000'
```

f-strings

- A string literal denoted with a `f` before the quotes can evaluate expressions placed within braces, including references to variables, function calls and comparisons
- This provides an expressive and concise way to define string objects

- It supported from Python 3.6+

```
name = "Langlands"
"The name {} has {} characters in it and the number of a's {}".format(name, len(name),name.count('a'))
```

```
"The name Langlands has 9 characters in it and the number of a's 2"
```

```
name = 'Langlands'
f'The name {name} has {len(name)} letters and {name.lower().count("a")} "a" s.'
```

```
'The name Langlands has 9 letters and 2 "a" s.'
```

```
Hyd_Bangalore = 635
Bangalore_Chennai = 462
total_distance = Hyd_Bangalore + Bangalore_Chennai
print(f"The total distance travelled: {total_distance} km")
```

```
The total distance travelled: 1097 km
```

```
mph = 60
time = total_distance / mph
print(f'Total time taken to travel {total_distance}  km is {round(time,3)}')
```

```
Total time taken to travel 1097  km is 18.283
```

Lists

- Python provides data structures for holding an *ordered* list of objects
- In languages like C and Fortran such data structure is called an *array* and it can hold only one type of data (for instance, an array of integers)
- The array structure in python can hold a mixture of data types
- A Python *list* is an ordered, mutable array of objects.
- A list is constructed by specifying the objects, separated by commas, between square brackets []
- Python list can contain references to any type of object: strings, the various types of numbers, built-in constants such as the boolean value True, and even other lists.

```
list1 = [1, 'two', 3.14, 0]
list1
```

```
[1, 'two', 3.14, 0]
```

```
type(list1)
```

```
list
```

```
a = 4
list2 = [2, a, -0.1, list1, True]
list2
```

```
[2, 4, -0.1, [1, 'two', 3.14, 0], True]
```

```
list2[-1][0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-23-536d9c6b619b> in <module>
----> 1 list2[-1][0]

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

SEARCH STACK OVERFLOW

```
# empty list
list0 = []
list0
```

```
[]
```

```
list2[3][1][2]
```

```
'o'
```

```
list2[3][1]
```

```
'two'
```

```
list2[3]
```

```
[1, 'two', 3.14, 0]
```

```
list2[3][1][:2]
```

```
'tw'
```

```
list2
```

```
[2, 4, -0.1, [1, 'two', 3.14, 0], True]
```

```
4 in list2
```

```
True
```

```
10 in list2
```

```
False
```

```
'two' in list2
```

```
False
```

```
'two' in list2[3]
```

```
True
```

```
q1 = [1, 2, 3]
q1[1] = 'two'
q1
```

```
[1, 'two', 3]
```

```
# Slicing the list
q1 = [0., 0.1, 0.2, 0.3, 0.4, 0.5,0.7,0.8]
q1[1:7:2]
```

```
[0.1, 0.3, 0.5]
```

```
q1[::-1] # return a reversedcopy of the list
```

```
[0.5, 0.4, 0.3, 0.2, 0.1, 0.0]
```

```
q1[1::3] # striding: returns element at 1, 3, 5
```

```
[0.1, 0.4, 0.8]
```

```
q2 = q1[1:4]
q2[1] = 99
q2
```

```
[0.1, 99, 0.3]
```

```
q1
```

```
[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
```

List Methods

- Python *list* come with a large number of useful methods.
- Since the *list* are mutable, they can grow or shrink in a place
- Some useful *list* methods
  - `append` - add an item to the end of the list
  - `extend` - add one or more objects by copying them from another list
  - `insert` - insert an item at a specified index;
  - `remove` - remove a specified item from the list.

- `reverse` - Reverse the list in place
- `sort` - Sort the list in place
- `copy` - Return a copy of the list
- `count` - Return the number of elements equal to element in the list
- `index` - Return the lowest index of the list containg element
- `pop` - Remove and return the last element from the list

```
q1.append?
```

```
Signature: q1.append(object, /)
Docstring: Append object to the end of the list.
Type:      builtin_function_or_method
```

```
q = []
q.append(4)
q
```

```
[4]
```

```
q.extend([6,7,9])
q
```

```
[4, 6, 7, 9]
```

```
q.insert(1,5) # insert 5 at index 1
q
```

```
[4, 5, 6, 7, 9]
```

```
q.remove(7)
```

```
q
```

```
[4, 5, 6, 9]
```

```
q.append(6)
q
```

```
[4, 5, 6, 9, 6]
```

```
q.index(6)
```

```
2
```

```
q.sort()
q
```

```
[4, 5, 6, 6, 9]
```

```
q.reverse()
q
```

```
[9, 6, 6, 5, 4]
```

```
q.pop()
q
```

```
['a', 'e', 'b', 'f', 'A']
```

```
q = ['a', 'e','b','f','A','X']
q.sort()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-54-de7980ddb74b> in <module>
      1 q = ['a', 'e','b','f','A','X']
----> 2 q.sorted()

AttributeError: 'list' object has no attribute 'sorted'
```

SEARCH STACK OVERFLOW

```
sorted(q)
```

```
['A', 'X', 'a', 'b', 'e', 'f']
```

```
q.sort()
q
```

```
['A', 'X', 'a', 'b', 'e', 'f']
```

> By default, `sort()` and `sorted()` order the items in an array in *ascending order*

```
sorted(q,reverse=True) # for descending order
```

```
['f', 'e', 'b', 'a', 'X', 'A']
```

```
q = [5, '4', 3, 2, 1]
sorted(q)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-55-582c5e7cbe92> in <module>
      1 q = [5, '4', 3, 2, 1]
----> 2 sorted(q)

TypeError: '<' not supported between instances of 'str' and 'int'

SEARCH STACK OVERFLOW
```

```
q.sort()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-131-27cdafc62d64> in <module>
----> 1 q.sort()

TypeError: '<' not supported between instances of 'str' and 'int'

SEARCH STACK OVERFLOW
```

Tuples

- A tuple can be thought as an immutable list.
- Tuples are constructed by placing the items inside the parentheses.
- Tuples can be *indexed* and *sliced* in the same way as *lists* but, being *immutable*, they cannot be *appended* to, *extended* or have elements removed from them.

```
t = (1,'two',3)
t
```

```
(1, 'two', 3)
```

```
type(t)
```

```
tuple
```

```
t[1]
```

```
'two'
```

```
t[2] = 4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-135-3feed840db06> in <module>
----> 1 t[2] = 4

TypeError: 'tuple' object does not support item assignment

SEARCH STACK OVERFLOW
```

```
t = (1, list1, 0)
t
```

```
(1, [1, 'two', 3.14, 0], 0)
```

```
t[1][0]=4
```

```
t
```

```
(1, [4, 'two', 3.14, 0], 0)
```

```
type(t[1])
```

```
list
```

```
t[1][1] = 'Three' # Ok to change the list within the tuple
t
```

```
(1, [1, 'Three', 3.14, 0], 0)
```

Disctionaries

- A *disctionay* in Python is a type of "associative array"
- A dictionary can contain any objects as its values, but unlike sequences such as lists and tuples, in which the items are indexed by an integer starting at 0, each item in a dictionary is indexed by a unique *key*, which may be any *immutable* object.
- The distcionry is a collection of *key-value* pairs
- Disctionaries are *mutable* objects
- The dictionary *keys* must be unique, the dictionary *values* need not be.

```
height = {'Burj Khalifa': 828., 'One World Trade Center': 541.3,
          'Mercury City Tower': -1., 'Q1':323.,
          'Eiffel Tower': 324}
```

```
height
```

```
{'Burj Khalifa': 828.0,
 'One World Trade Center': 541.3,
 'Mercury City Tower': -1.0,
 'Q1': 323.0,
 'Eiffel Tower': 324}
```

```
type(height)
```

```
dict
```

```
height[0]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-45-e35583b92c64> in <module>
----> 1 height[0]

KeyError: 0
```

SEARCH STACK OVERFLOW

```
height['Burj Khalifa']
#height[0]
```

```
828.0
```

```
building = 'Q1'
height[building]
```

```
323.0
```

```
height['The Shard'] = 306.0
height['Q1'] = 306.0
```

```
height
```

```
{'Burj Khalifa': 828.0,
 'One World Trade Center': 541.3,
 'Mercury City Tower': -1.0,
 'Q1': 306.0,
 'Eiffel Tower': 324,
 'The Shard': 306.0}
```

An alternative way of defining a dictionary is to pass a sequence of *(key, value)* pairs to the dict constructor.

```
ordinal = dict([(1,'First'),(2,'Second'),(3,'Third')])
mass = dict(Mercury=3.301e23, Venus=4.867e24, Earth=5.972e24)
mass
```

```
{'Mercury': 3.301e+23, 'Venus': 4.867e+24, 'Earth': 5.972e+24}
```

```
mass.items()
```

```
dict_items([('Mercury', 3.301e+23), ('Venus', 4.867e+24), ('Earth', 5.972e+24)])
```

```
mass['Earth']
```

```
5.972e+24
```

```
ordinal[2]
```

```
'Second'
```

Methods in Dictionary

Some useful dictionary methods.

- `get` - used to retrive the value, given a key if it exists
- `keys` - return a dictioney keys
- `values` - return a dictionary values
- `items` - return a dictionary items

```
dir(mass)
```

```
['__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__ror__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']
```

```
mass.items()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-40-660823adf82a> in <module>
----> 1 mass.items(2)

TypeError: dict.items() takes no arguments (1 given)
```

SEARCH STACK OVERFLOW

```
mass.keys()
```

```
dict_keys(['Mercury', 'Venus', 'Earth'])
```

```
planets = mass.keys()
planets
```

```
dict_keys(['Mercury', 'Venus', 'Earth'])
```

```
print(planets)
```

```
dict_keys(['Mercury', 'Venus', 'Earth'])
```

```
planet_list = list(mass.keys())
```

```
planet_list
```

```
['Mercury', 'Venus', 'Earth']
```

```
planet_list[0]
```

```
'Mercury'
```

Sets

- A `set` is an *unordered* collection of *unique items*
- A set is useful for removing duplicates from a sequence and for determining the union, intersection and difference between two collections.
- Since they are unordered, set objects cannot be indexed or sliced, but they can be iterated over, tested for membership and they support the `len` built-in.
- A `set` is created by listing its elements between braces ({...}) or set() constructor.

```
s = set([1,1,4,3,2,2,3,4,1,3,'Surprise!'])
s
```

```
{1, 2, 3, 4, 'Surprise!'}
```

```
type(s)
```

```
set
```

```
len(s)
```

```
5
```

```
2 in s
```

```
True
```

```
6 not in s
```

```
True
```

```
4 in s, 6 not in s
```

```
(True, True)
```

Methods in set

The following are some useful methods in set object

- `add` - to add elemts to given set
- `remove` - to remove a specified element but raises a *KeyError exception* if the element is not present in the set;
- `discard` - It does the same as remove but does not raise an error in this case.
- `pop` - removes an arbitrary element from the set
- `clear` -removes all elements from the set

```
s = {2, -2, 0}
s.add(1)
s
```

```
{-2, 0, 1, 2}
```

```
s.add(-1)
s.add(1.0)
s
```

```
{-2, -1, 0, 1, 2}
```

```
s.remove(3)
s
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-4-227c54533f3f> in <module>
----> 1 s.remove(3)
      2 s

KeyError: 3
```

SEARCH STACK OVERFLOW

```
s.discard(3)
s
```

```
{-2, -1, 0, 2}
```

```
s.remove(3)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-173-b06e0f5ff900> in <module>
----> 1 s.remove(3)

KeyError: 3
```

SEARCH STACK OVERFLOW

```
s.pop()
s
```

```
{-2, -1, 2}
```

```
s.clear()
s
```

```
set()
```

```
dir(s)
```

```
['__and__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__iand__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__isub__',
```

```
'__iter__',
'__ixor__',
'__le__',
'__len__',
'__lt__',
'__ne__',
'__new__',
'__or__',
'__rand__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__ror__',
'__rsub__',
'__rxor__',
'__setattr__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__xor__',
'add',
'clear',
'copy',
'difference',
'difference_update',
'discard',
'intersection',
'intersection_update',
'isdisjoint',
'issubset',
'issuperset',
'pop',
'remove',
'symmetric_difference',
'symmetric_difference_update',
'union',
'update']
```

Set methods for set operations

- union
- intersection
- difference
- symmetric_difference
- isdisjoint
- issubset
- issuperset

```
A = set([1,2,3])
B = set((1,2,3,4))
A <= B
```

```
True
```

```
(A.issubset(B)) == (A <= B)
```

```
True
```

```
C, D = set((3,4,5,6)), set((7,8,9))
# B or C # union
```

```
{1, 2, 3, 4}
```

```
B | C # union
```

```
{1, 2, 3, 4, 5, 6}
```

```
A | C | D # A U C U D
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
A & C # intersection
```

```
{3}
```

```
C & D
```

```
set()
```

```
C.isdisjoint(D)
```

```
True
```

```
B - C # difference
```

```
{1, 2}
```

```
B^C # symmetric difference
```

```
{1, 2, 5, 6}
```

```
(B - C) | (C- B)
```

```
{1, 2, 5, 6}
```

Frozensets

- sets are mutable objects
- A frozenset object is immutable set
- Frozensets are fixed, unordered collections of unique objects and can be used as dictionary keys and set members.

```
a = set((1,2,3))
b = set(('q',(1,2),a))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-a23517cb01f6> in <module>
      1 a = set((1,2,3))
----> 2 b = set(('q',(1,2),a))

TypeError: unhashable type: 'set'
```

SEARCH STACK OVERFLOW

```
a = frozenset((1,2,3))
b = set(('q',(1,2),a))
b
```

```
{(1, 2), frozenset({1, 2, 3}), 'q'}
```

```
dir(a)
```

```
['__and__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__rand__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__ror__',
 '__rsub__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
```

```
'__sub__',
'__subclasshook__',
'__xor__',
'copy',
'difference',
'intersection',
'isdisjoint',
'issubset',
'issuperset',
'symmetric_difference',
'union']
```

Questions for Revision

These review questions copied from Jovian page

Try answering the following questions to test your understanding of the topics covered in this notebook:

1. What is a variable in Python?
2. How do you create a variable?
3. How do you check the value within a variable?
4. How do you create multiple variables in a single statement?
5. How do you create multiple variables with the same value?
6. How do you change the value of a variable?
7. How do you reassign a variable by modifying the previous value?
8. What does the statement `counter += 4` do?
9. What are the rules for naming a variable?
10. Are variable names case-sensitive? Do `a_variable`, `A_Variable`, and `A_VARIABLE` represent the same variable or different ones?
11. What is Syntax? Why is it important?
12. What happens if you execute a statement with invalid syntax?
13. How do you check the data type of a variable?
14. What are the built-in data types in Python?
15. What is a primitive data type?
16. What are the primitive data types available in Python?
17. What is a data structure or container data type?
18. What are the container types available in Python?
19. What kind of data does the Integer data type represent?
20. What are the numerical limits of the integer data type?
21. What kind of data does the float data type represent?
22. How does Python decide if a given number is a float or an integer?
23. How can you create a variable which stores a whole number, e.g., 4 but has the float data type?
24. How do you create floats representing very large (e.g., 6.023 x 10^23) or very small numbers (0.000000123)?
25. What does the expression `23e-12` represent?
26. Can floats be used to store numbers with unlimited precision?
27. What are the differences between integers and floats?
28. How do you convert an integer to a float?
29. How do you convert a float to an integer?
30. What is the result obtained when you convert 1.99 to an integer?
31. What are the data types of the results of the division operators `/` and `//`?
32. What kind of data does the Boolean data type represent?
33. Which types of Python operators return booleans as a result?
34. What happens if you try to use a boolean in arithmetic operation?
35. How can any value in Python be covered to a boolean?
36. What are truthy and falsy values?
37. What are the values in Python that evaluate to False?
38. Give some examples of values that evaluate to True.
39. What kind of data does the None data type represent?
40. What is the purpose of None?
41. What kind of data does the String data type represent?
42. What are the different ways of creating strings in Python?
43. What is the difference between strings creating using single quotes, i.e. `'` and `'` vs. those created using double quotes, i.e. `"` and `"`?
44. How do you create multi-line strings in Python?
45. What is the newline character, `\n`?
46. What are escaped characters? How are they useful?
47. How do you check the length of a string?

48. How do you convert a string into a list of characters?
49. How do you access a specific character from a string?
50. How do you access a range of characters from a string?
51. How do you check if a specific character occurs in a string?
52. How do you check if a smaller string occurs within a bigger string?
53. How do you join two or more strings?
54. What are "methods" in Python? How are they different from functions?
55. What do the `.lower`, `.upper` and `.capitalize` methods on strings do?
56. How do you replace a specific part of a string with something else?
57. How do you split the string "Sun,Mon,Tue,Wed,Thu,Fri,Sat" into a list of days?
58. How do you remove whitespace from the beginning and end of a string?
59. What is the string `.format` method used for? Can you give an example?
60. What are the benefits of using the `.format` method instead of string concatenation?
61. How do you convert a value of another type to a string?
62. How do you check if two strings have the same value?
63. Where can you find the list of all the methods supported by strings?
64. What is a list in Python?
65. How do you create a list?
66. Can a Python list contain values of different data types?
67. Can a list contain another list as an element within it?
68. Can you create a list without any values?
69. How do you check the length of a list in Python?
70. How do you retrieve a value from a list?
71. What is the smallest and largest index you can use to access elements from a list containing five elements?
72. What happens if you try to access an index equal to or larger than the size of a list?
73. What happens if you try to access a negative index within a list?
74. How do you access a range of elements from a list?
75. How many elements does the list returned by the expression `a_list[2:5]` contain?
76. What do the ranges `a_list[:2]` and `a_list[2:]` represent?
77. How do you change the item stored at a specific index within a list?
78. How do you insert a new item at the beginning, middle, or end of a list?
79. How do you remove an item from al list?
80. How do you remove the item at a given index from a list?
81. How do you check if a list contains a value?
82. How do you combine two or most lists to create a larger list?
83. How do you create a copy of a list?
84. Does the expression `a_new_list = a_list` create a copy of the list `a_list`?
85. Where can you find the list of all the methods supported by lists?
86. What is a Tuple in Python?
87. How is a tuple different from a list?
88. Can you add or remove elements in a tuple?
89. How do you create a tuple with just one element?
90. How do you convert a tuple to a list and vice versa?
91. What are the `count` and `index` method of a Tuple used for?
92. What is a dictionary in Python?
93. How do you create a dictionary?
94. What are keys and values?
95. How do you access the value associated with a specific key in a dictionary?
96. What happens if you try to access the value for a key that doesn't exist in a dictionary?
97. What is the `.get` method of a dictionary used for?
98. How do you change the value associated with a key in a dictionary?
99. How do you add or remove a key-value pair in a dictionary?
100. How do you access the keys, values, and key-value pairs within a dictionary?