Data Structures and Lab

Built-In Data types(structures) in Python

Muhammad Tariq Mahmood

tariq@koreatech.ac.kr
School of Computer Science and Engineering

Note: These notes are prepared from the following resources.

- Starting Out with Python, Pearson by Tony Gaddis (2021)
- Introduction to Programming Using Python, Pearson by Y. Daniel Liang, .
- https://docs.oracle.com/javase/tutorial/ (tutorials, and references).
- https://www3.ntu.edu.sg/home/ehchua/programming/index.html#Java
- https://docs.python.org/3/tutorial/

Contents

1 Built-In Data types(Data Structures) in Python

- String
- Tuple
- List
- Tuple
- Dictionary
- Set

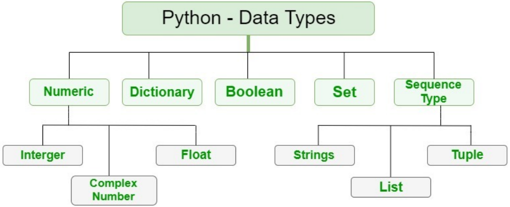
2 Files in Python

- Reading txt Files in Python
- Writing txt Files in Python
- Reading and Writing CSV Files in Python

Built-In Data types(Data Structures) in Python

Built-In Data types(Data Structures) in Python

- Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data.
- Python has a large number of built-in data types, such as Numbers (Integer, Float, Boolean, Complex Number), String, List, Tuple, Set, Dictionary and File.
- More high-level data types, such as Decimal and Fraction, are supported by external modules.
- Python associates types with objects, instead of variables. That is, a variable does not have a fixed type and can be assigned an object of any type. A variable simply provides a reference to an object.
- Data types are actually classes and variables are instances (objects) created from these classes.

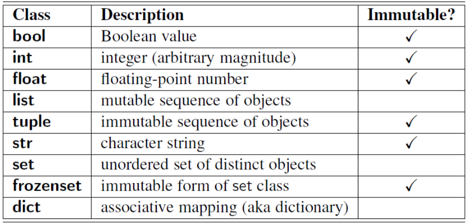


- A class is **immutable** if each object of that class has a fixed value upon instantiation that cannot subsequently be changed.

Built-In Data types(Data Structures) in Python (cont...)

Classes for data types

- bool: It is used to manipulate logical (Boolean) values: True and False.
- int: It is designed to represent integer values with arbitrary magnitude.
- float: It is the sole floating-point type in Python, using a fixed-precision representation.
- list: A list instance stores a sequence of objects.
- tuple: It provides an **immutable version** of a sequence. While Python uses the [ ] characters to delimit a list, parentheses delimit a tuple, with () being an empty tuple.
- str: It is specifically designed to efficiently represent an immutable sequence of characters.
- set: It represents the mathematical notion of a set, namely a **collection of elements**, without duplicates, and without an inherent order to those elements.
- dict: It represents a **dictionary, or mapping**, from a set of distinct **keys** to associated **values**.



Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

- Python's classes may also define one or more **methods** (also known as **member functions**), which are invoked on a specific instance of a class using the **dot (".")** operator.

Built-In Data types(Data Structures) in Python (cont...)

Data Structures for single item

- Integers (type int): e.g., 123, -456. Unlike C/C++/Java, integers are of unlimited size in Python.

```
x=234
print(x)
print(type(123)) # <class 'int'>
```

- Floating-point numbers (type float): e.g., 1.0, -2.3, 3.4e5, -3.4E-5. floats are 64-bit double precision floating-point numbers

```
x=2.34
print(x)
print(type(x)) #<class 'float'>
```

- Booleans (type bool): takes a value of either True or False.

```
x=True
print(x)
print(type(x)) # <class 'bool'>
print(bool(0))   # Cast int 0 to bool - False
print(bool(1))   # Cast int 1 to bool - True
print(bool([])  # Cast empty list to bool - False
print(bool([1,2,3])  # Cast empty list to bool - True
```

- Complex Numbers (type complex): e.g., 1+2j, -3-4j. Complex numbers have a real part and an imaginary part denoted with suffix of j (or J).

```
x = 1 + 2j
print(x) # (1+2j)
print(x.real)  # 1
print(type(x)) # <class 'complex'>
print(x * (3 + 4j)) # (-5+10j)
```

- Other number types are provided by external modules, such as decimal module for decimal fixed-point numbers, fraction module for rational numbers.

Built-In Data types(Data Structures) in Python (cont...)

```
import decimal  # Using the decimal module
x = decimal.Decimal('0.1')  # Construct a Decimal object
x * 3    # Multiply with overloaded * operator
print(type(x))  # Get type <class 'decimal.Decimal'>
```

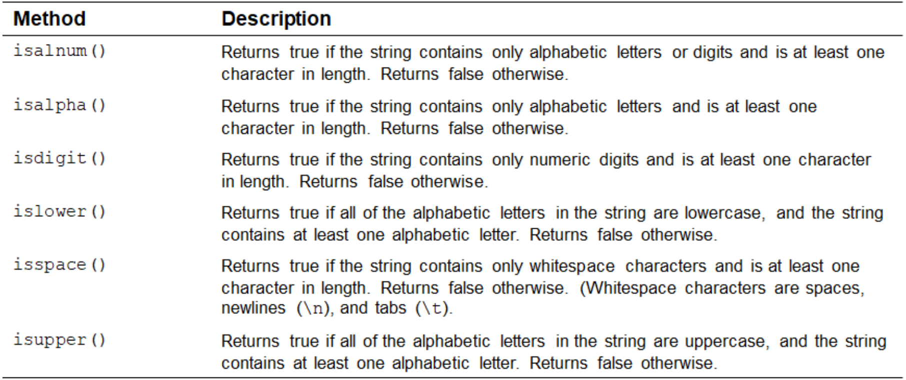
Data Structures for multiple items

- Strings are Immutable: Strings are immutable, i.e., their contents cannot be modified. String functions such as upper(), replace() returns a new string object instead of modifying the string under operation.
- List: [v1, v2, ...] (mutable dynamic array).
- Tuple: (v1, v2, v3, ...) (Immutable fix-sized array).
- Dictionary: {k1:v1, k2:v2, ...} (mutable key-value pairs, associative array, map).
- Set: {k1, k2, ...} (with unique key and mutable).

Python String

Python String

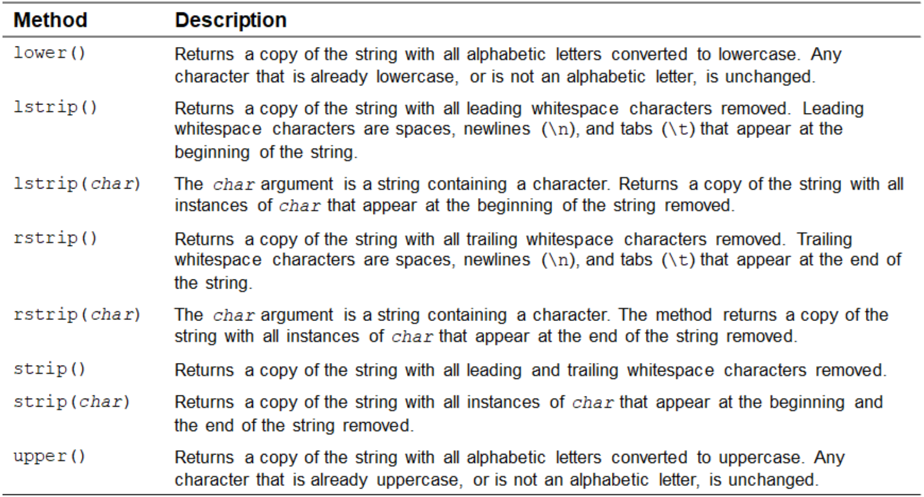
- In Python, strings can be delimited by a pair of single-quotes ('...') or double-quotes ("..."). Python also supports multi-line strings via triple-single-quotes ("'..."') or triple-double-quotes ("""...""").
- Python provides several ways to access the individual characters in a string. Strings also have methods that allow you to perform operations on them.
- String Testing Methods: The string methods test a string for specific characteristics



Method	Description
`isalnum()`	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
`isalpha()`	Returns true if the string contains only alphabetic letters and is at least one character in length. Returns false otherwise.
`isdigit()`	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
`islower()`	Returns true if all of the alphabetic letters in the string are lowercase, and the string contains at least one alphabetic letter. Returns false otherwise.
`isspace()`	Returns true if the string contains only whitespace characters and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (`\n`), and tabs (`\t`).
`isupper()`	Returns true if all of the alphabetic letters in the string are uppercase, and the string contains at least one alphabetic letter. Returns false otherwise.

Python String (cont...)

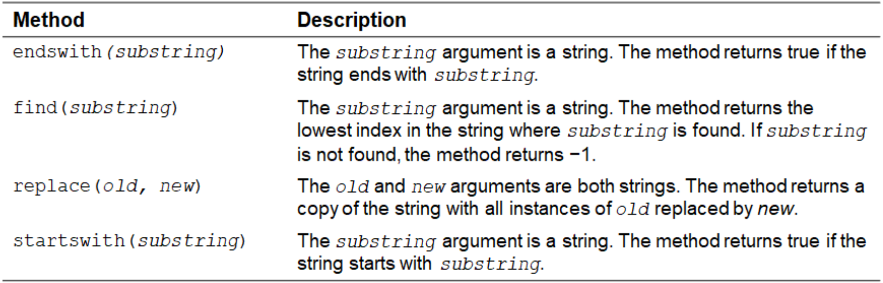
- Modification Methods: Strings are immutable, i.e., their contents cannot be modified. String functions such as upper(), replace() returns a new string object instead of modifying the string under operation.



Method	Description
`lower()`	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
`lstrip()`	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the beginning of the string.
`lstrip(`*`char`*`)`	The *`char`* argument is a string containing a character. Returns a copy of the string with all instances of *`char`* that appear at the beginning of the string removed.
`rstrip()`	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the end of the string.
`rstrip(`*`char`*`)`	The *`char`* argument is a string containing a character. The method returns a copy of the string with all instances of *`char`* that appear at the end of the string removed.
`strip()`	Returns a copy of the string with all leading and trailing whitespace characters removed.
`strip(`*`char`*`)`	Returns a copy of the string with all instances of *`char`* that appear at the beginning and the end of the string removed.
`upper()`	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.

Python String (cont...)

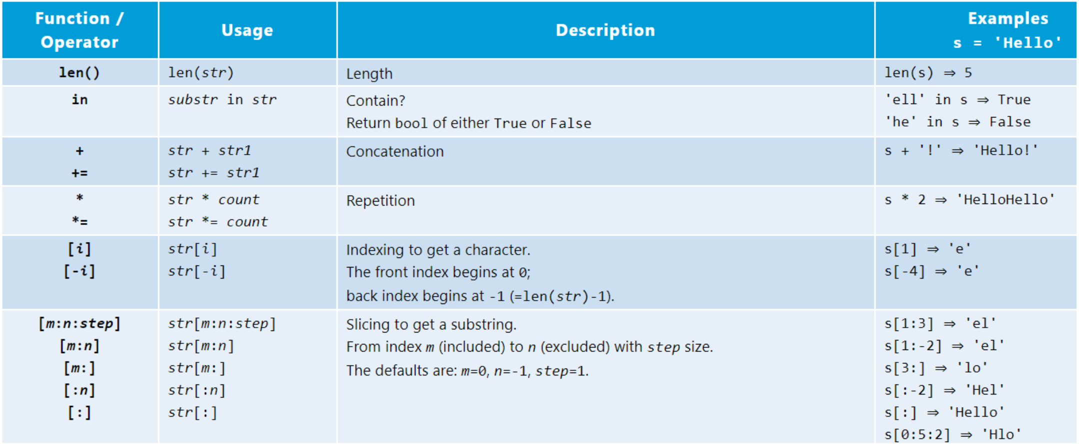
- Searching and Replacing: Programs commonly need to search for substrings, or strings that appear within other strings.
- Table lists some of the Python string methods that search for substrings, as well as a method that replaces the occurrences of a substring with another string.



Method	Description
endswith *(substring)*	The *substring* argument is a string. The method returns true if the string ends with *substring*.
find(*substring*)	The *substring* argument is a string. The method returns the lowest index in the string where *substring* is found. If *substring* is not found, the method returns −1.
replace(*old, new*)	The *old* and *new* arguments are both strings. The method returns a copy of the string with all instances of *old* replaced by *new*.
startswith(*substring*)	The *substring* argument is a string. The method returns true if the string starts with *substring*.

Python String (cont...)

- Character Type? Python does not have a dedicated character data type. A character is simply a string of length 1. You can use the indexing operator to extract individual character from a string. The built-in functions ord() and chr() operate on character.
- Python provides several ways to access the individual characters in a string. Summary of String operations



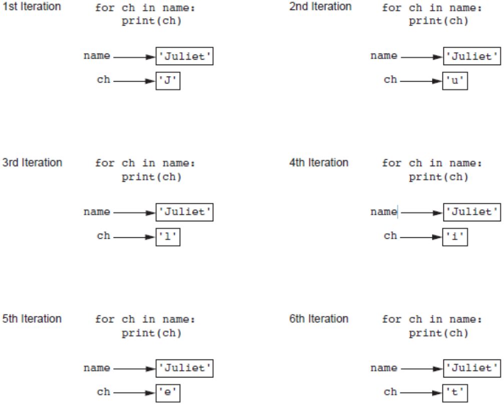
Function / Operator	Usage	Description	Examples s = 'Hello'
len()	len(*str*)	Length	len(s) ⇒ 5
in	*substr* in *str*	Contain? Return bool of either True or False	'ell' in s ⇒ True 'he' in s ⇒ False
+ +=	*str* + *str1* *str* += *str1*	Concatenation	s + '!' ⇒ 'Hello!'
* *=	*str* * *count* *str* *= *count*	Repetition	s * 2 ⇒ 'HelloHello'
[*i*] [-*i*]	*str*[*i*] *str*[-*i*]	Indexing to get a character. The front index begins at 0; back index begins at -1 (=len(*str*)-1).	s[1] ⇒ 'e' s[-4] ⇒ 'e'
[*m*:*n*:*step*] [*m*:*n*] [*m*:] [:*n*] [:]	*str*[*m*:*n*:*step*] *str*[*m*:*n*] *str*[*m*:] *str*[:*n*] *str*[:]	Slicing to get a substring. From index *m* (included) to *n* (excluded) with *step* size. The defaults are: *m*=0, *n*=-1, *step*=1.	s[1:3] ⇒ 'el' s[1:-2] ⇒ 'el' s[3:] ⇒ 'lo' s[:-2] ⇒ 'Hel' s[:] ⇒ 'Hello' s[0:5:2] ⇒ 'Hlo'

Python String (cont...)

- Examples of String usage

```
st = 'Hello, World!'   # single quotes
print(st)
st = "Hello, World!"   # double quotes
print(st)
st= """String literals can
span multiple lines."""
print(st)
print(type(s))
print(dir(s)) # List all attributes of the object s

#The str() Constructor
st=str(234)
st=str(2+5j)
st = str('ABCDEFGHI')
print(st)

print(st[0])     # Prints A
print(st[4])     # Prints E
print(st[-1])     # Prints I
print(st[-6])     # Prints D

#slicing
print(st[2:5])       # Prints CDE
print(st[5:-1])      # Prints FGH
print(st[1:6:2])     # Prints BDF

st[0] = 'J' # error String is not mutable
print(st)
```

```
name = 'Juliet'
for ch in name:
     ch = 'X'
     print(name)
```



Python Tuple

Tuple (v1, v2,...)

- Tuple is similar to list except that it is immutable (just like string). Hence, tuple is more efficient than list. A tuple consists of items separated by commas, enclosed in parentheses ().

- The parentheses are actually optional, but recommended for readability. Nevertheless, the commas are mandatory. For example,

```python
tup1 = (5,)  # An one-item tuple needs a comma
tup2 = 123, 4.5, 'hello' #
tup = (123, 4.5, 'hello')
print(tup[1])
print(tup[1:3])
tup[1] = 9       # Tuple, unlike list, is immutable
print(type(tup))
print(lst = list(tup))  # Convert to list)
```

- You can operate on tuples using (supposing that tup is a tuple):
  - built-in functions such as len(tup);
  - built-in functions for tuple of numbers such as max(tup), min(tup) and sum(tup);
  - operators such as in, + and *; and
  - tuple's member functions such as tup.count(item), tup.index(item), etc.

Python List

List

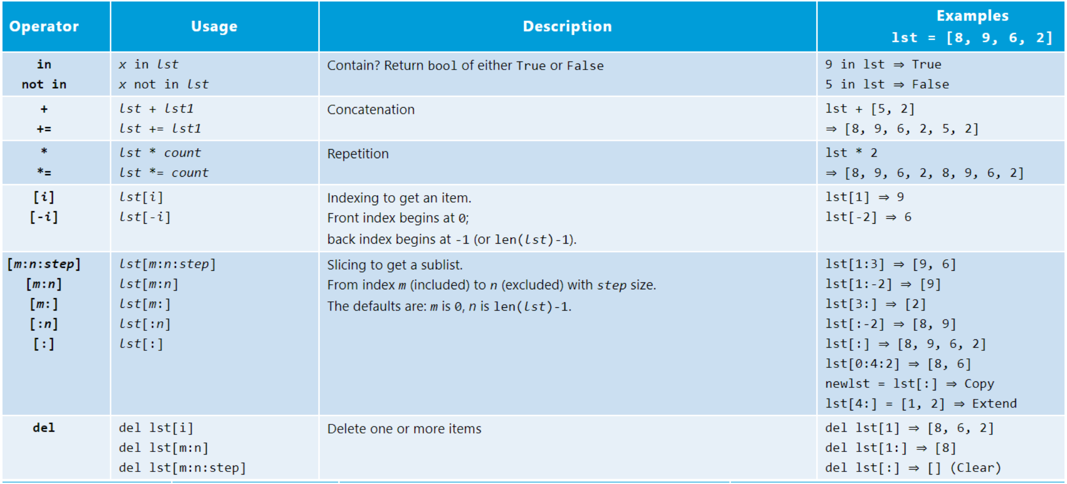
- A list is a sequence of values (similar to an array in other programming languages but more versatile)

- The values in a list are called items or sometimes elements.

- The important properties of Python lists are as follows:
  - A list is enclosed by square brackets [].
  - A list can contain items of different types. It is because Python associates types to objects, not variables.
  - A list grows and shrinks in size automatically (dynamically). You do not have to specify its size during initialization.
  - list, unlike string, is mutable. You can insert, remove and modify its items.
  - You can index the items from the front with positive index, or from the back with negative index. E.g., if lst is a list, lst[0] and lst[1] refer to its first and second items; lst[-1] and lst[-2] refer to the last and second-to-last items.
  - You can also refer to a sub-list (or slice) using slice notation lst[m:n:step] (from index m (included) to index n (excluded) with step size).

Python List (cont...)

- Summary of list operations



Operator	Usage	Description	Examples `lst = [8, 9, 6, 2]`

`in`
`not in`
	`x in lst`
	`x not in lst`
		Contain? Return bool of either `True` or `False`	`9 in lst ⇒ True`
			`5 in lst ⇒ False`

`+`
`+=`
	`lst + lst1`
	`lst += lst1`
		Concatenation	`lst + [5, 2]`
			`⇒ [8, 9, 6, 2, 5, 2]`

`*`
`*=`
	`lst * count`
	`lst *= count`
		Repetition	`lst * 2`
			`⇒ [8, 9, 6, 2, 8, 9, 6, 2]`

`[i]`
`[-i]`
	`lst[i]`
	`lst[-i]`
		Indexing to get an item.
		Front index begins at `0`;
		back index begins at `-1` (or `len(lst)-1`).
			`lst[1] ⇒ 9`
			`lst[-2] ⇒ 6`

`[m:n:step]`
`[m:n]`
`[m:]`
`[:n]`
`[:]`
	`lst[m:n:step]`
	`lst[m:n]`
	`lst[m:]`
	`lst[:n]`
	`lst[:]`
		Slicing to get a sublist.
		From index *m* (included) to *n* (excluded) with *step* size.
		The defaults are: *m* is `0`, *n* is `len(lst)-1`.
			`lst[1:3] ⇒ [9, 6]`
			`lst[1:-2] ⇒ [9]`
			`lst[3:] ⇒ [2]`
			`lst[:-2] ⇒ [8, 9]`
			`lst[:] ⇒ [8, 9, 6, 2]`
			`lst[0:4:2] ⇒ [8, 6]`
			`newlst = lst[:] ⇒ Copy`
			`lst[4:] = [1, 2] ⇒ Extend`

`del`	`del lst[i]`
	`del lst[m:n]`
	`del lst[m:n:step]`
		Delete one or more items	`del lst[1] ⇒ [8, 6, 2]`
			`del lst[1:] ⇒ [8]`
			`del lst[:] ⇒ [] (Clear)`

Python List (cont...)

Operations and issues to be associated with lists

- Properties of Lists
- Length of Lists
- Slicing with Lists
- Printing elements of Lists with Loops
- Access to Lists via Indexes
- Adding element(s) to the Lists
- Concating Lists
- Change on List elements
- Deleting element(s) from the Lists
- Finding elements in Lists
- Copying a List
- Some Operations with Lists (sort, reverse, min-max, sum)
- Nested Lists

Python List (cont...)

list-Specific Member Functions : The list class provides many member functions. Suppose *lst* is a list object:

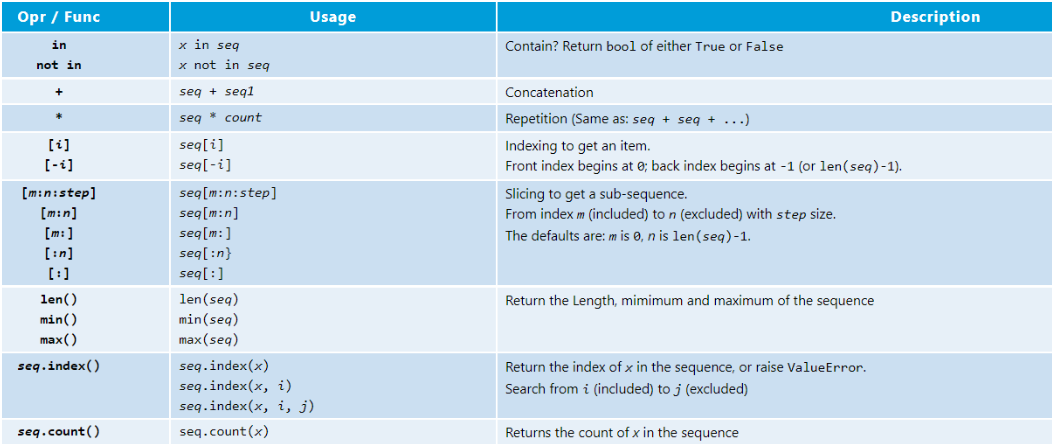
- lst.index(item): return the index of the first occurrence of item; or error.
- lst.append(item): append the given item behind the lst and return None.
- lst.extend(lst1): append the given list lst1 behind the lst and return None
- lst.insert(index, item): insert the given item before the index and return None. Hence,
- lst.insert(0, item) inserts before the first item of the lst; lst.insert(len(lst), item) inserts at the end of the lst which is the same as lst.append(item).
- lst.remove(item): remove the first occurrence of item from the lst and return None; or error.
- lst.pop(): remove and return the last item of the lst.
- lst.pop(index): remove and return the indexed item of the lst.
- lst.clear(): remove all the items from the lst and return None; same as operator del lst[:].
- lst.count(item): return the occurrences of item.
- lst.reverse(): reverse the lst in place and return None.
- lst.sort(): sort the lst in place and return None.
- lst.copy(): return a copy of lst; same as lst[:].
- Examples of String usage

Python List (cont...)

```python
list1 = ['hello', 'how', 'are', 'you', 1, 10, 'how', 'well', 'are'] # created a simple list
print(list1)
print(len(list1))  # length of the list1
print(list1[0]) # first element
print(list1[8]) # last element

print(list1[-9]) # first element
print(list1[-1]) # last element
# slicing
print(list1[:3])  # left-closed type data retrieval
print(list1[2:5])  # 2-4 are retrived
print(list1[2:])  # 2-8 elements are retrieved.
print(list1[:4]) # 0-3 elements.
print(list1[-1:])  # -1 implies last element of the list
print (list1[:-1]) # only the last element is skipped
print(list1[:-2]) # last and last-before elements are skipped (-1 and -2 respectively)
print(list1[::-1])  # to reverse the order
# Deleting list
del list1[4] # the list1[4] was deleted and replaced by the next element in the list
print(list1)
#Replacing elements
list1[4] = 1232
print(list1)
# inserting element at position 4
list1.insert(4, 111)
```

Python List (cont...)

- list, tuple, and str are parts of the sequence types. list is mutable, while tuple and str are immutable. They share the common sequence's built-in operators and built-in functions.



Opr / Func	Usage	Description

`in`
`not in`
	x `in` *seq*
	x `not in` *seq*
		Contain? Return bool of either `True` or `False`

`+`	*seq* `+` *seq1*	Concatenation
`*`	*seq* `*` *count*	Repetition (Same as: *seq* `+` *seq* `+ ...`)

`[`*i*`]`
`[-`*i*`]`
	seq`[`*i*`]`
	seq`[-`*i*`]`
		Indexing to get an item.
		Front index begins at `0`; back index begins at `-1` (or `len(`*seq*`)-1`).

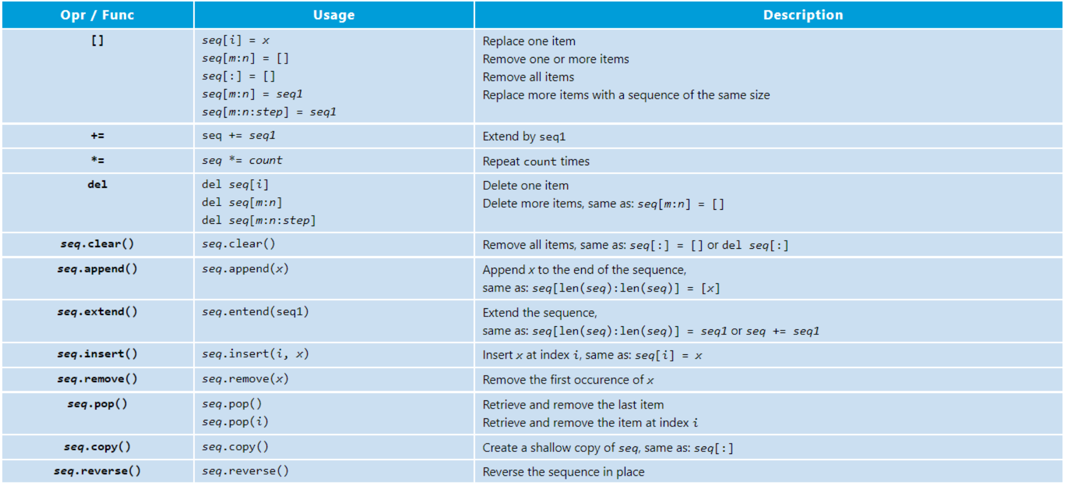
`[`*m*`:`*n*`:`*step*`]`
`[`*m*`:`*n*`]`
`[`*m*`:]`
`[:`*n*`]`
`[:]`
	seq`[`*m*`:`*n*`:`*step*`]`
	seq`[`*m*`:`*n*`]`
	seq`[`*m*`:]`
	seq`[:`*n*`}`
	seq`[:]`
		Slicing to get a sub-sequence.
		From index *m* (included) to *n* (excluded) with *step* size.
		The defaults are: *m* is `0`, *n* is `len(`*seq*`)-1`.

`len()`
`min()`
`max()`
	`len(`*seq*`)`
	`min(`*seq*`)`
	`max(`*seq*`)`
		Return the Length, mimimum and maximum of the sequence

seq`.index()`	*seq*`.index(`*x*`)` *seq*`.index(`*x*`,` *i*`)` *seq*`.index(`*x*`,` *i*`,` *j*`)`	Return the index of *x* in the sequence, or raise `ValueError`. Search from *i* (included) to *j* (excluded)
seq`.count()`	`seq.count(`*x*`)`	Returns the count of *x* in the sequence

Python List (cont...)

- For mutable sequences (list), the following built-in operators and built-in functions (func(seq)) and member functions (seq.func(*args)) are supported:



Opr / Func	Usage	Description
[]	seq[i] = x seq[m:n] = [] seq[:] = [] seq[m:n] = seq1 seq[m:n:step] = seq1	Replace one item Remove one or more items Remove all items Replace more items with a sequence of the same size
+=	seq += seq1	Extend by seq1
*=	seq *= count	Repeat count times
del	del seq[i] del seq[m:n] del seq[m:n:step]	Delete one item Delete more items, same as: seq[m:n] = []
seq.clear()	seq.clear()	Remove all items, same as: seq[:] = [] or del seq[:]
seq.append()	seq.append(x)	Append x to the end of the sequence, same as: seq[len(seq):len(seq)] = [x]
seq.extend()	seq.entend(seq1)	Extend the sequence, same as: seq[len(seq):len(seq)] = seq1 or seq += seq1
seq.insert()	seq.insert(i, x)	Insert x at index i, same as: seq[i] = x
seq.remove()	seq.remove(x)	Remove the first occurence of x

seq.pop()	seq.pop()
	seq.pop(i)
		Retrieve and remove the last item
		Retrieve and remove the item at index i

seq.copy()	seq.copy()	Create a shallow copy of seq, same as: seq[:]
seq.reverse()	seq.reverse()	Reverse the sequence in place

Tuple

Tuple (v1, v2,...)

- In Python, Tuples are a data structure of the sequence type that store a **collection of data**.
- Tuples are ordered – Tuples maintains a left-to-right positional ordering among the items they contain.
- Accessed by index – Items in a tuple can be accessed using an index.
- Tuples can contain any sort of object – It can be numbers, strings, lists and even other tuples.
- Tuples are immutable – you can't add, delete, or change items after the tuple is defined.
- Creating tuples

```python
#Creating tuples
t0=() # Empty Tuple
t1=(1,) # Tuple with a single value
t2= (1, 2, 3) # Tuple containing numeric objects
t3=('hello', 'world') # Tuple containing string objects
t4=(True, [1, 2], (3, 4), 'hello') # Tuple containing multiple objects
# Converting list() and tuple()
t5 = tuple([1, 2, 3])
print(tuple(['cat', 'dog', 5]))
# Tuple from dictionary
d = dict(a=1, b=2, c=3)
t6 = tuple(d)
```

Tuple (cont...)

Python Tuple Methods

- all() – Return true if all elements of tuples are true or tuple is empty
- any() – Return true if any elements of tuples are true and False when tuple is empty
- enumerate() – Return an enumerate object from the tuple
- len() – Return the length of the tuple
- max() – Return the maximum value from the tuple
- min() – Return the minimum value from the tuple
- sum() – Return the sum of all values of the tuple
- sorted() – Return a sorted list of the values of the tuple
- tuple() – Converts a sequence to a tuple

Dictionary

Dictionary {k1:v1, k2:v2,...}

- Python's built-in dictionary type supports key-value pairs (also known as name-value pairs, associative array, or mappings).
- A dictionary is enclosed by a pair of curly braces . The key and value are separated by a colon (:), in the form of {k1:v1, k2:v2, ...}
- Unlike list and tuple, which index items using an integer index 0, 1, 2, 3,..., dictionary can be indexed using any key type, including number, string or other types.
- Dictionary is mutable.
- Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.
- a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'.
- Examples

```python
# Lets store age of people as dictionary
dict1 = {"Julie": 32, "Rahul": 23, "Jasmine": 12, "Jack": 15, "Jennifer": 18}
print(dict1)
print(dict1['Jasmine']) # To retrieve values
print(dict1.keys()) # To get list of keys
print(dict1.values()) # To get list of values

dict1= dict([('Julie', 32), ('Rahul', 23), ('Jasmine', 12)]) # method 2 to create dict
print(dict1)
dict1 = dict(Julie=32, Rahul=23, Jasmine=12) # method 3 to create dict
print(dict1)
print(type(dict1))

dict1={x: 4*x for x in range(1,5)} # method 4 - list comprehension technique
print(dict1)
print(type(dict1))

#printing dict
for k, v in dict1.items():
print(k, v)

dictReversed = {v:k for k, v in dict1.items()}  # Note the curly braces
print(dictReversed)
```

Dictionary (cont...)

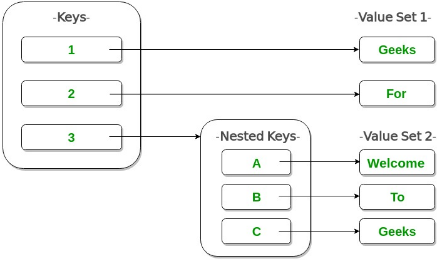
Operations and issues to be associated with dictionaries

- Properties of Dictionaries
- Creating a Dictionary
- Access to Dictionary Element(s)
- Access with Loops
- Adding element(s) to the Dictionaries
- Deleting element(s) from the Dictionaries
- Checking for the existence of a key
- Merging two Dictionaries
- Copying a Dictionary
- Nested Dictionaries

Dictionary (cont...)

- Creating a Nested Dictionary

```
Dict = {1: 'Geeks', 2: 'For',
3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Geeks'}}
```



- Accessing an element of a nested dictionary

```
Dict = {'Dict1': {1: 'Geeks'},'Dict2': {'Name': 'For'}}
# Accessing element using keys
print(Dict['Dict1'])
print(Dict['Dict1'][1])
print(Dict['Dict2']['Name'])
```

- Adding elements to a Dictionary

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Adding elements one at a time
Dict[0] = 'Geeks'
Dict[2] = 'For'
Dict[3] = 1
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Adding set of values
# to a single Key
Dict['Value_set'] = 2, 3, 4
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[2] = 'Welcome'
print("\nUpdated key value: ")
print(Dict)

# Adding Nested Key value to Dictionary
Dict[5] = {'Nested' :{'1' : 'Life', '2' : 'Geeks'}}
print("\nAdding a Nested Key: ")
print(Dict)
```

Dictionary (cont...)

- Dictionary-Specific Member Functions : The dict class has many member methods. The commonly-used are follows (suppose that dct is a dict object):
  - dct.has_key(): Returns true if key in dictionary dict, false otherwise
  - dct.items(), dct.keys(), dct.values(): returns a lists of items, keys and values in a given dictionary.
  - dct.clear(): The clear() method removes all items from the dictionary.
  - dct.copy(): They copy() method returns a shallow copy of the dictionary.
  - dct.get(): It is a conventional method to access a value for a key.
  - dct.update(dct2): merge the given dictionary dct2 into dct. Override the value if key exists, else, add new key-value.
  - dct.pop(): Removes and returns an element from a dictionary having the given key.

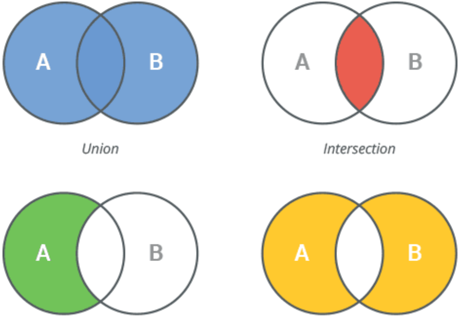
- Examples

```
dct = {'name':'Peter', 'age':22, 'gender':'male'}
print(type(dct))  # Show type <class 'dict'>
print(dir(dct))   # Show all attributes of dct object
print(list(dct.keys()))       # Get all the keys as a list
print(list(dct.values()))     # Get all the values as a list
print(list(dct.items()))      # Get key-value as tuples
print(dct.get('age', 'not such key'))  # Retrieve item
print(dct['height'])   # Indexing an invalid key raises KeyError, while get() could gracefully handle invalid key
del dct['age']   # Delete (Remove) an item of the given key
print('name' in dct) #True
dct.update({'height':180, 'weight':75})  # Merge the given dictionary
dct.pop('gender')  # Remove and return the item with the given key
dct.pop('no_such_key')   # Raise KeyError if key not found
dct.pop('no_such_key', 'not found')   # Provide a default if key does not exist 'not found'
```

Set

Set {k1, k2,...}

- The important properties of Python sets are as follows:
  - Set elements can not be accessed by index and they are unordered. (The order will be different each time you want to access sets.)
  - They are immutable like tuples. Differences between sets and tuples are, we can add and remove new elements to the sets but we cannot change an element in the set!
  - Sets are defined with (curly brackets or braces). Elements in the sets are separated by commas.
  - They can contain different types of values.
  - They can NOT contain duplicate element which means two same element that has same value.



Union

Intersection

Difference

Symmetric Difference

- Set-Specific Operators : Python supports set operators & (intersection), | (union), (difference) and ˆ(exclusive-or).

Set (cont...)

- Examples

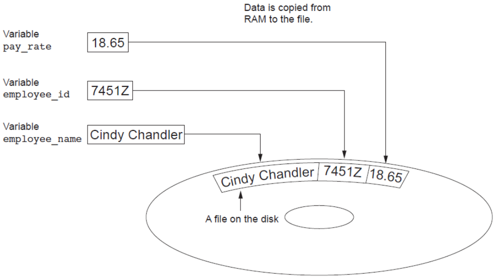
```python
st = {123, 4.5, 'hello', 123, 'Hello'}
print(st)
print(88 in st)
st2 = set([2, 1, 3, 1, 3, 2])
print(st2)
st3 = set('hellllo')
print(st3)
list1 = ['hello', 'how', 'are', 'you', 1, 10, 'how', 'well', 'are']
set1 = set(list1)
print(set1)

set1 = {'hello', 'how', 'are', 1, 1232, 'well'}
print(set1)

# set1[2] = 'lel' will not work as the values cannot be replaced
set1.add('lel')  # values can be added - dynamic
print(set1)
```

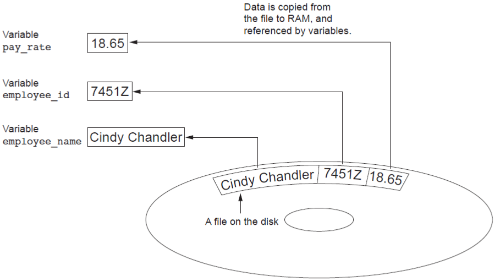
```python
st3 = set('hellllo')
print(st3)
st4 = {'a', 'e', 'i', 'o', 'u'}
print(st3 | st4)
print(st3 & st4)
print(st3 - st4)
print(st3 ^ st2)
```

Files in Python

Files in Python

- When a program needs to save data for later use, it writes the data in a file. The data can be read from the file at a later time.
- Programs that are used in daily business operations rely extensively on files. Payroll programs keep employee data in files, inventory programs keep data about a company's products in files, accounting systems keep data about a company's financial operations in files, and so on.
- The process of retrieving data from a file is known as "reading data from" the file.





- In general, there are two types of files: text and binary.

Files in Python (cont...)

- Here are some of the functions in Python that allow you to read and write to files:
  - read() : This function reads the entire file and returns a string
  - readline() : This function reads lines from that file and returns as a string. It fetch the line n, if it is been called nth time.
  - readlines() : This function returns a list where each element is single line of that file.
  - write() : This function writes a fixed sequence of characters to a file.
  - writelines() : This function writes a list of string.
  - append() : This function append string to the file instead of overwriting the file.

Files in Python (cont...)

Reading txt Files in Python

- Python provides a wide range of built-in functions for file handling. It makes it really easy to create, update, read, and delete files.

- Open a File: You can open a file using open() built-in function specifying its name.

```
f = open('myfile.txt')  # Open a file for reading
f = open('myfile.txt', 'r')
f = open('myfile.txt', 'w')  # Open a file for writing, (overwrite)
f = open('myfile.txt', 'a')   # Open a file for writing, (append)
f = open('myfile.txt', 'r+')  # Open a file for reading and writing
f = open('myfile.txt', 'rb') # Open a binary file for reading
```

- Read a File: To read its contents, you can use read() method. By default, the read() method reads the entire file. However, you can specify the maximum number of characters to read.

```
f = open('myfile.txt')
print(f.read())  # read entire file
print(f.read(3)) # read first 3 characters
print(f.read(5)) # read first 5 characters
```

- Read Lines : To read a single line from the file, use readline() method. If you want to read all the lines in a file into a list of strings, use readlines() method.

```
f = open('myfile.txt')
print(f.readline()) # Prints First line of the file.
print(f.readline()) # Call it again to read next line, Prints Second line of the file.
print(f.readlines()) # Read all the lines in a file into a list of strings
#You can loop through an entire file line-by-line using a simple for loop.
f = open('myfile.txt')
for line in f:
    print(line)
```

Files in Python (cont...)

Writing txt Files in Python

- Write a File: Use the write() built-in method to write to an existing file. Remember that you need to open the file in one of the writing modes (‘w’, ‘a’ or ‘r+’) first.

```python
f = open('myfile.txt', 'w')
f.write('Overwrite existing data.')
f.write(' Append this text.')

#To write multiple lines to a file at once, use writelines() method. This method accepts list of strings as an input.


lines = ['New line 1\n', 'New line 2\n', 'New line 3']
f.writelines(lines)
f.flush() # Flush output buffer to disk without closing
```

- Close a File : Use the close() function to close an open file.

```python
# Method-1
f = open('myfile.txt')
f.close()

# check closed status
print(f.closed)
# Prints True

# Method-2
with open('myfile.txt') as f:
    print(f.read())

^^I^^I
# Method-3
f = open('myfile.txt')
try:
    # File operations goes here
finally:
    f.close()
```

CSV Files in Python

Reading and Writing CSV Files in Python

- A CSV file (Comma Separated Values file) is a delimited text file that uses a comma , to separate values. It is used to store tabular data, such as a spreadsheet or database.

- Python's Built-in **csv library** makes it easy to read, write, and process data from and to CSV files.

- Open a CSV File: You can open a file using open() built-in function specifying its name (same as a text file).

```python
f = open('myfile.csv') # Open a file for reading
f = open('myfile.csv', 'w') # Open a file for writing
f = open('myfile.csv', 'r+') # Open a file for reading and writing
```

- Close a CSV File : use the close() function to close an open file.

```python

#method 01
f = open('myfile.csv')
f.close()
# check closed status
print(f.closed) # Prints True


#method 02
with open('myfile.csv') as f:
    print(f.read())



#method 03
f = open('myfile.csv')
try:
# File operations goes here
finally:
f.close()
```

CSV Files in Python (cont...)

- Read a CSV File You can read its contents by importing the csv module and using its reader() method. The reader() method splits each row on a specified delimiter and returns the list of strings.

```python
import csv

with open('myfile.csv') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

- Write to a CSV File : To write an existing file, you must first open the file in one of the writing modes (‘w’, ‘a’ or ‘r+’) first. Then, use writerow() method.

```python
import csv

with open('myfile.csv', 'w') as f:
   writer = csv.writer(f)
   writer.writerow(['Bob', '25', 'Manager', 'Seattle'])
   writer.writerow(['Sam', '30', 'Developer', 'New York'])
```