Summary Slides for
Chapters 1 to 9
And
basic data structures (List, Tuple, Set, Dictionary)

Chapter 1: Introduction

Computer
- executes a set of instructions
- IPO (Input, Processing, Output)

Interpreter vs Compiler
- An interpreter runs the code right away
- An interpreter stops the execution when it hits an error
- The compiler translates source code into machine code first.
- The compiler can do this only if the source code is error-free.
- The computer then runs the machine code.

Language
- alphabet (a, b, c)
- lexis aka dictionary
- syntax (Alex went to school – ok; Alex school to went – not ok)
- semantics (School went to Alex – not ok)

Types of Languages
- Natural Language (what we speak)
- Programming Language (What humans use to interact with the machines)
- Low-level Machine Language (Assembly Language)
- High-Level Programming Language (Python, Java, C, JS, C#)

Types of Code
- Pseudo Code (sequence of steps written in plain English)
- Source Code (code written in a high-level programming language)
- Binary (aka machine) code (Meant for the computers)

Python
- Is a high-level programming language
- Is an interpreter
- Is very simple, and powerful in scripting, testing, etc.
- Uses C language underneath
- Cpython is a reference implementation.

Chapter 2 Input, Processing and Output (IPO)

PYTHON PROGRAMMING by SIVA JASTHI

2. Input, Processing, Output

Basic Data Types
- int float str bool

Advanced Data Types
- list tuple set dict

Python Key Words (33)
- Value Keywords: True, False, None
- Operator Keywords: and, or, not, in, is
- Control Flow Keywords: if, elif, else
- Iteration Keywords: for, while, break, continue, else
- Structure Keywords: def, class, with, as, pass, lambda
- Returning Keywords: return, yield
- Import Keywords: import, from, as

Constants
- Values do not change
- Use all UPPER CASE (INTEREST_RATE)

Variables
- Values do change
- Use lowercase and pot hole case (student_name)
- Can't use Python keywords or spaces or symbols

Variables: Three things matter
- 1. name, 2. value, 3. data type -
- type(var) is your friend to know the data type
- print(var) is your friend to know its value

Python goes from TOP to BOTTOM
- A = 10; B = 20
- C = A + B # OK
- X = A + B + D # Not OK

Python goes from RIGHT to LEFT during the assignment
- A = 2
- A = A + 1
- A = A * A

Assignment
- a = 10 (means "Assign the value of 10 to a"
- Short-hand assignment operator
  - a = 2
  - a += 2 # a = a + 2
  - a *= a # a = a * a

Basic Functions
- input( ) to get inputs. Returns a string
- type( ) for knowing the type of a variable
- print( ) to display outputs to console
  - print(*args, sep = ' ', end = '\n')
  - Can print many arguments
  - Can print different data types
  - \n new line character (escape char)
  - \t tab character (escape char)

Data Conversion (Casting)
- From str to int - int("2")
- From int to str: - str(2)
- From str to float: float('2.3')
- From str to Boolean: bool('True')

Arithmetic Operators
- Addition (+) Subtraction (-)
- Multiplication (*) Modulus (%)
- Division
- Float Division (/)
- Floor Division (//) (integer; goes to small)
- Exponentiation (**)

Operator Precedence
- PEMDAS or GEMS (Group, Expo, Multi, Sub)
- If in doubt, throw in a parenthesis
- ** right binding; rest is left binding

Coding Conventions
- Commenting
- Empty lines to separate the blocks of code
- Consistency

Chapter 3: Conditions

3. Conditions:

Boolean Logic
- True
- False

Comparison Operators

>	Greater Than
<	Less Than
>=	Greater Than or Equal sTo
<=	Less Than or Equals To
==	Equals To
!=	Not Equal To

Logical Operators
- and (every operand must be True)
- or (one of the operands must be True)
- not (negation)

Truth Tables
- True and True → True
- True and False → False
- False and True → False
- False and False → False

- True or True → True
- True or False → True
- False or True → True
- False or False → False

- not(True) → False
- not(False) → True

1 and 0 reflect True and False
- 0 is always False
- 1 is always True
- Non-zero is also True

Short-hand evaluation
- And case (stops when expr is false)
- Or case (stops when expr is true)
- a = 100 and 1 # a = 1
- b = 0 and 100 # b = 0
- c = 100 or 1 # c = 100
- d = 0 or 1 # c = 1

Membership Operators
- in
- not in (inverse of in)

Conditions
- Python goes from top to bottom
- Conditions control the flow
- if
- if..else
- if. elif…else
- if..elif…elif..elif..else
- Nested conditions
    - If
        - If
        - Else
    - Else
- Nested Conditions can be replaced with logical operators

Chapter 4: Iterations (Loops) (range, for, while)

Random number generation
- radom.randint(a, b) # between a and b
- radom.radint( ) # between 0 and 1

while loops:
- Indefinite loops
- Keep doing something until the condition is True
- break – break out of the loop (break can only appear inside a loop)
- continue - stop and go back to the start of the loop

range function
- range(start, stop, step)
- range(start, stop)   # step = 1 by default
- range(stop) #start = 0; step = 1 by default

for loops:
- for in range(start, stop, step)
- for in range(start, stop)
- for in range(stop)
- for in sequence (* this will be covered In ch7)
    - Sequence = list, tuple, set (*), dict, range
- break – breaks out of for loop
- continue – goes to the start of the loop

Sentinel in loops:
- Sentinel = a guard whose job is to keep watch.
- While or for Loops: use sentinel to break out of the loop

Extending for and while loops:
- We can have a “else” block associated with while or for loops
- “else” block is executed if the loop executes normally (without any interruption / without any “break”)

Ch5: Functions

- Functions ( )
  - Building blocks help code reuse
  - Tested and proven
  - Use it and take their services for granted
  - Analogy: Calling Uber; Withdrawing Money from Bank

- Defining Functions (aka Signature)
  - def keyword
  - Fuction has a name
  - Arguments (aka parameters)
  - Can take 0, 1 or N inputs
  - Can return 0, 1 or N outputs

- Calling (aka Invoking) Functions (calling UBER)
  - Position (order) of arguments matter
  - If there is a mismatch, you will get an error

- Return values
  - Functions **may** return a value
  - If they return a value, you use it.
  - Ignoring a return value is not good
  - Using a return value of **None** is also not good
  - Control exits a function upon hitting the first return

- Void-Returning Functions
  - No need to specify a return
  - returns **None** by default
- Value-Returning Functions
  - If you forget to add a return, the function returns None by default
- Positional Arguments
  - Binding between the caller and the function based on the position of the argument
  - Number of arguments matter
  - Position of arguments matter

- Optional Parameters
  - Use = to assign default value to the arguments
  - That argument then becomes optional
  - Optional arguments can only be in the end
  - How does python think?
    - If you pass me a value, I will use yours.
    - If you don't, I will use mine.

- Keyword Arguments
  - You can call methods through keywords
  - Order (position) of arguments doesn't matter
  - You can call a method
    - either through position
    - or through keywords
  - You can force the keywords through *
    - def method1(a, b, *, c, d)
      - Positional: a, b Keywords: c, d
    - def method1(*, a, b, c, d)
      - Positional: NA Keywords: a,b, c, d
-
- Pass-By-Value (int, float, str, bool) (scalar variables)
  - Functions only get a copy
  - What you do inside will not impact the original

- Pass-By-Value (list, tuple, set, dict)
  - A copy of the remote is passed
  - However, the remote is still pointing to the original
  - Any change you do through 'copy of the remote' will impact the original TV as well

- Scope of variables (LEGB)
  - LG (Local vs Global) Redefining / Shadowing a global
  - Local (Function) scope: You can overwrite the local scope with "global" keyword
  - Enclosing (inner function) scope
  - Global scope
  - Built-in Scope

- Variable Number of Arguments
  - Varargs (*args)
  - Prefix the argument with *
  - def total(*nums)
  - The above can be invoked as:
    - x = total(1,2)
    - x = total(3,4,5)
    - s = total(4,5,6,7)
- Variable Number of Key Word arguments
  - kwargs (**args)

- Built-in vs User-Defined Functions
  - built-in → print() input() len() sum()
  - User-defined: anything we write

- Functions vs Methods
  - Functions = built-in functions
  - Functions are not owned by anyone
    - len(some_list)
    - len(some_dict)
    - int(some_string)
    - int(some_float)
  - Methods are owned by some data type
    - my_list.append(2)
    - my_list.remove(4)
    - set_1.union(set_2)

- Abstraction
  - We only care about what the function does
  - We do NOT care about how it does!
  - We care about
    - name
    - arguments (inputs)
    - What it returns (outputs)

Ch.6.1 Files

6. Files (not in PCEP)

Type of files
- Text files (ASCII files, humans can read)
- Binary files (jpeg, png, machines can read)

Path
- Relative Path (gasprices.txt; data\gasprices.txt)
- Absolute Path (C:\users\srj\MyDocuments\gasprices.txt)

The common pattern in file IO
* Open the file **(r - read,** w – write, a- append, x - create)
- Process the file
  - read() : read the entire file in one go
  - read(n): read n chars at one time
  - readline( ): Read one line at a time
  - readlines( ): Read all lines into a list
- Close the file

with WITH construct
- Resource management is done by python.
- No need to explicitly close the file
- with open(FILE_NAME) as quotes_file_obj:
-

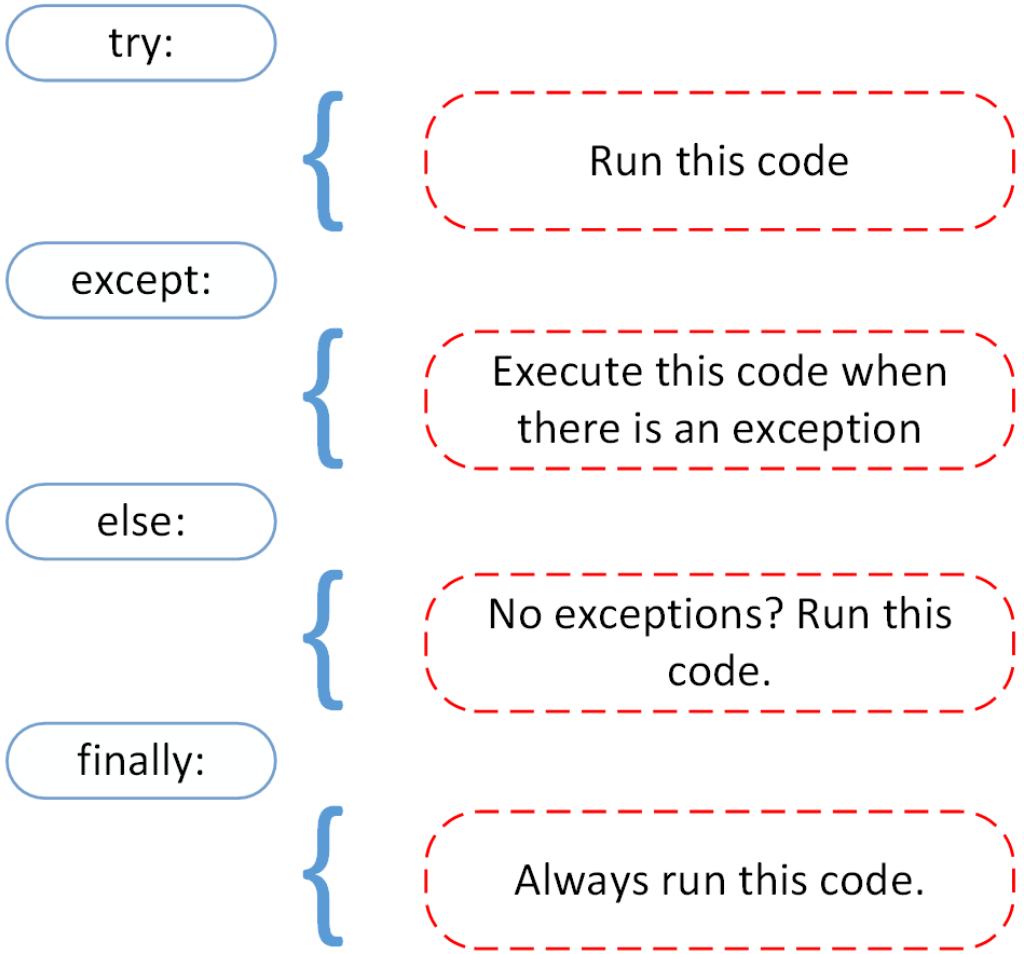
Without WITH construct:
- Programmer is responsible for closing the file
- file_name = "quotes.txt" #for humans
- file_obj = open(file_name) #file_obj is for python
- file_obj.close() # you must close the file yourself

Things can go wrong! Handle the Exceptions
- try Run this code
- except If something goes wrong, catch it
- else If there is nothing wrong, run this block
- finally Run this block regardless.

Ch.6.2 Exceptions

Things can go wrong! Handle the Exceptions

- try Run this code
- except If something goes wrong, catch it
- else If there is nothing wrong, run this block
- finally Run this block regardless.



Chapter 7 (List and Tuples) Summary

PYTHON PROGRAMMING by SIVA JASTHI

7. Lists and Tuples

Lists: An introduction
- Ordered, can contain duplicates, can be changed.
- Elements are accessed through index notation.
- Index starts from 0 to N-1; Supports negative indexing
- Can contain items of different data types ['alex', 17, 100.56]

List: Methods (you need to have a list to call these methods)

append(x)	extend(list_2)	insert(i,x)	remove(x)
pop()	clear()	index(x)	count(x)
sort()	reverse()	copy()	

High-Level abstraction of List operations
- Map (mapping each value of the list to something else)
- Filter (Creating a new list by filtering some items)
- Reduce (Summarizing the list data – count, length, max, min)

Traversal (for visiting each element in the list)
- for elem in range (we care about index)
- for elem in sequence (we care about values)
- For index, elem in enumerate(sequence) (we care about both)

Membership Operators:
- Is x in the list a? (x in a) or (x not in a)

List Unpacking:
- Assign the list elements to the variables
- my_list = ['red', 'green', 'blue', 'black']
- c1, c2, c3, c4 = my_list
- c1, *c2, cx = my_list
- You can also print an unpacked list by prefixing *
  - print(my_list) # printing the regular list
  - print(*my_list) # printing the unpacked list

List Slicing:
- Slices are sub-lists [start : stop : step]
- Defaults: start = 0; stop = N—1; step = 1
- Forward: 0, 1, 2, ….. N-1 Backward: -1, -2, -3, …. -N

List slicing examples:
- my_list = [10, 4, 5, 8, 9, 89]
- s1 = my_list[1 : 5]
- s2 = my_list[1 : 5 : 2]
- s3 = my_list[: 4]
- s4 = my_list[:]
- s5 = my_list[3 :]
- s6 = my_list[3 : -1]
- s7 = my_list[: :-1]
- s8 = my_list[: :]

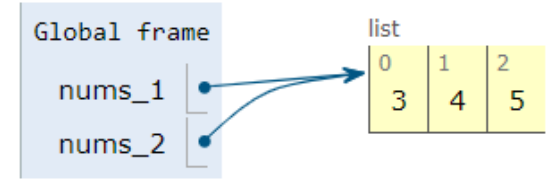
List Comprehension:
- Creating new lists from existing lists
- Syntax: [output, collection, condition]
- Example: [x+1 for x in range(10) if x%2==0]
- Condition is optional

Nested Lists:
- Lists containing other lists
- Can go any number of levels.

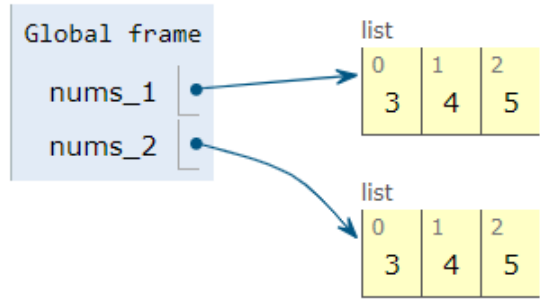
2-D Lists:
- Need two indexes [row] [column]
- Row-major and Column-major traversal

Tuples:
- Ordered, can contain duplicates
- Tuples are immutable (can't be modified)
- Tuple Methods: count( ) and index( )
- One item tuple = (10,) # Notice the comma
- Tuples can be added to create new tuples
  - a = (1,)
  - a = a + a

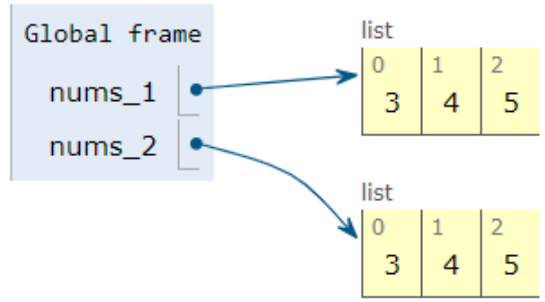
nums_1 = [3, 4, 5]
nums_2 = nums_1



nums_1 = [3, 4, 5]
nums_2 = nums_1.copy( )



nums_1 = [3, 4, 5]
nums_2 = nums_1[:]



Ch.8: More about Strings

1. string = character list

2. we use subscript notation
x[0], x[1] and so on.

3. you can use **for loop** to iterate a string

4. You get IndexError when you go out of bounds

5. len = length of a list, length of a string

6. You can add strings using + sign

7. Strings can NOT be changed.
Strings are immutable
name = 'Brown'
name[0] = 'C' --> You can not change a string

8. String formatting:
   - Strings can be formatted using f'' strings and { }

9. String slicing
string[:]
string[a:b] slice from a to b
string[a:] slide from a to the end of the list
string[:b] slice from beginning till b
string[-a:] slice from a spots from the end till the end

10. in or not in
are useful to check the presence of a sub-string

name = "John Hopkins"
x = 'J' in name
y = 'X' in name
z = 'X' not in name

11. Many string methods
→ For querying, searching
→ For modification, replacing
→ For splitting / tokenizing

12. Repetition Operator (*)

Ch 9: Sets and Dictionaries

Sets

- Not ordered, Not indexed
- Can not contain duplicates
- Supports Set operations
  - Union (|)
  - Intersection (&)
  - Differece (-)
  - Symmetric Difference (^)

- You can traverse the set using for loop
- CRUD operations
  - add(),
  - remove(),
  - discard(),
  - pop(),
  - update()

Dictionaries

- Not ordered, Not indexed
- Supports Key-Value pairs
- Can not contain duplicate keys

- Key can by any “immutable” data type
- Value can be anything

- CRUD Operations:
  - Create → setdefault( ), { }
  - Read → keys(), values(), items(), get()
  - Update → update( ), dictionary[key] = ??
  - Delete → pop( ), popitem( )

Misc. topics (Binary Representation)

- Binary representation
- Octal Representation
- Hex Representation

- Binary to decimal
- Decimal to binary
- Bitwise Operators
    - Bitwise OR |
    - Bitwise AND &
    - Bitwise XOR ^
    - Bitwise NOT !
    - Left Shift <<
    - Right Shift >>

Built-in Functions

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

https://www.w3schools.com/python/python_ref_functions.asp

Python Keywords

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

https://www.w3schools.com/python/python_ref_keywords.asp

Comparison of Data Structures Methods

	Lists	Tuples	Set	Dictionary
Ordered	✓	✓	✗	✗
Indexed	✓	✓	✗	✗
Add or Update items	✓	✗	✓	✓
Can contain duplicates	✓	✓	✗	✗
Supports Keys (Name: Values)	✗	✗	✗	✓
Uses	Square Brackets	Round Brackets	Curly Brackets	Curly Brackets
	[ ] list()	**( ) tuple()**	**set ()**	**{ } dict( )**

List Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

count( ) and index( ) Are only two methods applicable to tuples.

https://www.w3schools.com/python/python_lists_methods.asp

https://www.w3schools.com/python/python_ref_tuple.asp

Tuples Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

count( ) and index( ) Are only two methods applicable to tuples.

https://www.w3schools.com/python/python_ref_tuple.asp

Strings Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier

Strings are character collections.

https://www.w3schools.com/python/python_ref_string.asp

Strings Methods (Contd.)

islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list

Strings are character collections.

swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

https://www.w3schools.com/python/python_ref_string.asp

Set Methods

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two or more sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with another set, or any other iterable

https://www.w3schools.com/python/python_ref_set.asp

Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

https://www.w3schools.com/python/python_ref_dictionary.asp

Thank You.