PowerShell is called Terminal in the book.

Changing directory of PowerShell:- *Set-Location C:\Users\sohail.ahmad\Desktop\LPTHW\dir*

See files in a powershell directory:- *Get-ChildItem*

Run a python file from powershell:- *python ex1.py*

If you want to run Python in PowerShell, just type:- *python*

What is the order of operations? In the United States we use an acronym called PEMDAS which stands for Parentheses Exponents Multiplication Division Addition Subtraction. That's the order Python follows as well. The mistake people make with PEMDAS is to think this is a strict order, as in ”Do P, then E, then M, then D, then A, then S.” The actual order is you do the multiplication and division (M&D) in one step, from left to right, then you do the addition and subtraction in one step from left to right. So, you could rewrite PEMDAS as PE(M&D)(A&S).

Exercise 5: More Variables and Printing

Now we'll do even more typing of variables and printing them out. This time we'll use something called a ”format string.” Every time you put " (double-quotes) around a piece of text you have been making a string. A string is how you make something that your program might give to a human. You print strings, save strings to files, send strings to web servers, and many other things. Strings are really handy, so in this exercise you will learn how to make strings that have variables embedded in them. You embed variables inside a string by using a special {} sequence and then put the variable you want inside the {} characters. You also must start the string with the letter f for ”format”, as in f"Hello {somevar}". This little f before the " (double-quote) and the {} characters tell Python 3, ”Hey, this string needs to be formatted. Put these variables in there.”

```
my_name = 'Zed A. Shaw'

print(f"Let's talk about {my_name}.")

>> Let's talk about Zed A. Shaw.
```

Exercise 10: Escape Sequences

This is all of the escape sequences Python supports. You may not use many of these, but memorize their format and what they do anyway. Try them out in some strings to see if you can make them work. Foe example:- “\\” would help you writing single backslash (\).

Escape	What it does.
`\\`	Backslash (\)
`\'`	Single-quote (')
`\"`	Double-quote (")
`\a`	ASCII bell (BEL)
`\b`	ASCII backspace (BS)
`\f`	ASCII formfeed (FF)
`\n`	ASCII linefeed (LF)
`\N{name}`	Character named name in the Unicode database (Unicode only)
`\r`	Carriage Return (CR)
`\t`	Horizontal Tab (TAB)
`\uxxxx`	Character with 16-bit hex value xxxx
`\Uxxxxxxxx`	Character with 32-bit hex value xxxxxxxx
`\v`	ASCII vertical tab (VT)
`\ooo`	Character with octal value ooo
`\xhh`	Character with hex value hh

Exercise 11

We put a end=' ' at the end of each print line. This tells print to not end the line with a newline character and go to the next line. Also, input() function in python3 asks user input and then print it. In python2, it is raw_input(). When we give the input 22, it would then print "*How old are you? 22*"

print("How old are you?", end = ' ')

age = input()

INPUT AND ARGV

When we command line and want user to give an input then we use argv function. When we use script and want user to give the input, then we input() function inside the python script.

EXERCISE 16: READING AND WRITING FILES

If you did the Study Drills from the last exercise, you should have seen all sorts of commands (methods/functions) you can give to files. Here's the list of commands I want you to remember:

- close – Closes the file. Like File->Save.. in your editor.

- read – Reads the contents of the file. You can assign the result to a variable.
- readline – Reads just one line of a text file.
- truncate – Empties the file. Watch out if you care about the file.
- write('stuff') – Writes ”stuff” to the file.
- seek(0) – Move the read/write location to the beginning of the file.

EXERCISE 28: BOOLEAN LOGIC

Whenever you see these Boolean logic statements, you can solve them easily by this simple process:

1. Find an equality test (== or !=) and replace it with its truth.
2. Find each and/or inside parentheses and solve those first.
3. Find each not and invert it.
4. Find any remaining and/or and solve it.
5. When you are done you should have True or False.

EXERCISE 36: DESIGNING AND DEBUGGING

Rules for If-Statements

1. Every if-statement must have an else.
2. If this else should never run because it doesn’t make sense, then you must use a die function in the else that prints out an error message and dies, just like we did in the last exercise. This will find many errors.
3. Never nest if-statements more than two deep and always try to do them one deep.
4. Treat if-statements like paragraphs, where each if-elif-else grouping is like a set of sentences. Put blank lines before and after.
5. Your Boolean tests should be simple. If they are complex, move their calculations to variables earlier in your function and use a good name for the variable.

Rules for Loops

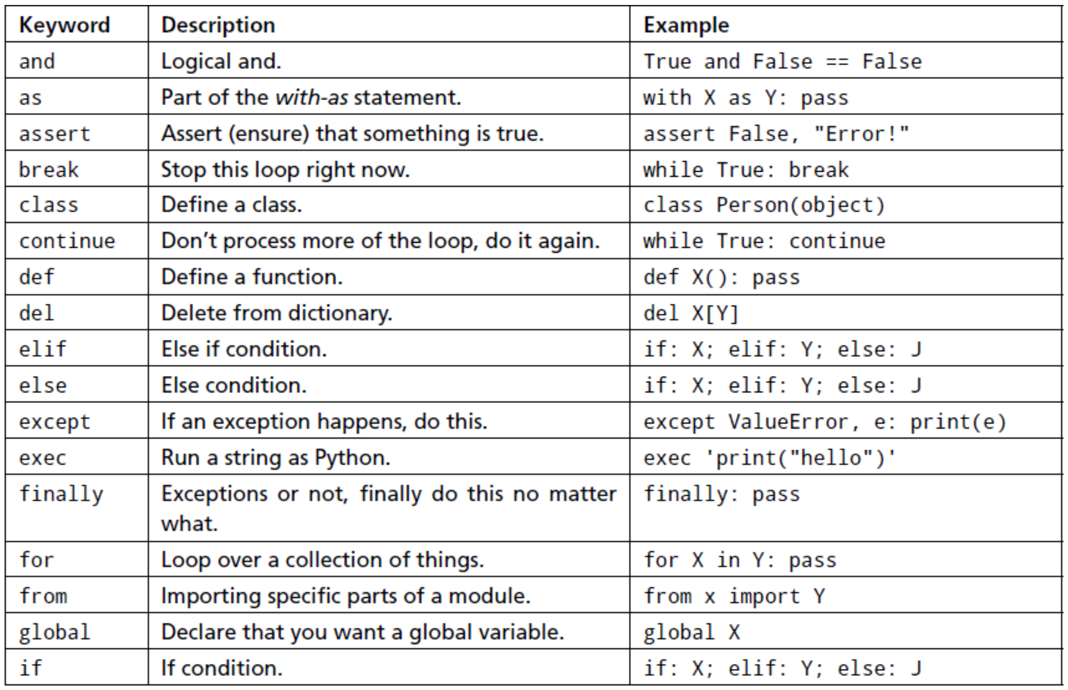
1. Use a while-loop only to loop forever, and that means probably never. This only applies to Python; other languages are different.
2. Use a for-loop for all other kinds of looping, especially if there is a fixed or limited number of things to loop over.

Tips for Debugging

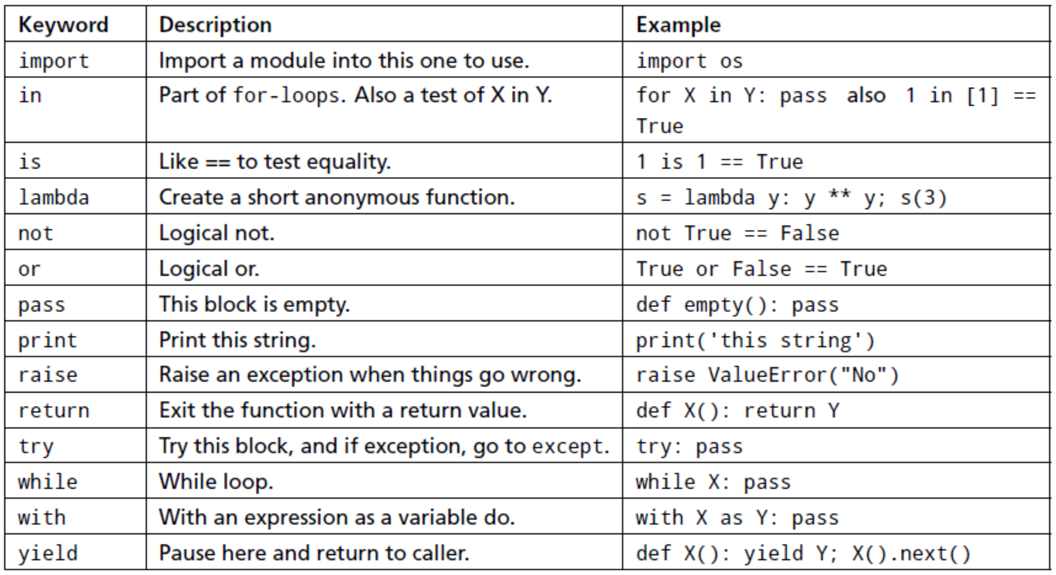
1. Do not use a ”debugger.” A debugger is like doing a full-body scan on a sick person. You do not get any specific useful information, and you find a whole lot of information that doesn’t help and is just confusing.
2. The best way to debug a program is to use print to print out the values of variables at points in the program to see where they go wrong.
3. Make sure parts of your programs work as you work on them. Do not write massive files of code before you try to run them. Code a little, run a little, fix a little.

EXERCISE 37: SYMBOL REVIEW

Python Keywords: Their description and example

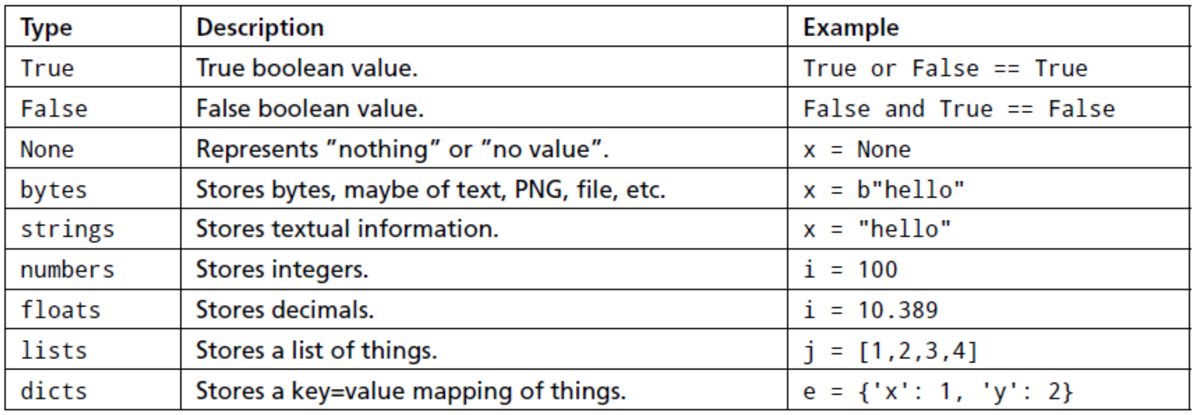


Keyword	Description	Example
`and`	Logical and.	`True and False == False`
`as`	Part of the *with-as* statement.	`with X as Y: pass`
`assert`	Assert (ensure) that something is true.	`assert False, "Error!"`
`break`	Stop this loop right now.	`while True: break`
`class`	Define a class.	`class Person(object)`
`continue`	Don’t process more of the loop, do it again.	`while True: continue`
`def`	Define a function.	`def X(): pass`
`del`	Delete from dictionary.	`del X[Y]`
`elif`	Else if condition.	`if: X; elif: Y; else: J`
`else`	Else condition.	`if: X; elif: Y; else: J`
`except`	If an exception happens, do this.	`except ValueError, e: print(e)`
`exec`	Run a string as Python.	`exec 'print("hello")'`
`finally`	Exceptions or not, finally do this no matter what.	`finally: pass`
`for`	Loop over a collection of things.	`for X in Y: pass`
`from`	Importing specific parts of a module.	`from x import Y`
`global`	Declare that you want a global variable.	`global X`
`if`	If condition.	`if: X; elif: Y; else: J`



Keyword	Description	Example
`import`	Import a module into this one to use.	`import os`
`in`	Part of `for-loops`. Also a test of X in Y.	`for X in Y: pass` also `1 in [1] == True`
`is`	Like == to test equality.	`1 is 1 == True`
`lambda`	Create a short anonymous function.	`s = lambda y: y ** y; s(3)`
`not`	Logical not.	`not True == False`
`or`	Logical or.	`True or False == True`
`pass`	This block is empty.	`def empty(): pass`
`print`	Print this string.	`print('this string')`
`raise`	Raise an exception when things go wrong.	`raise ValueError("No")`
`return`	Exit the function with a return value.	`def X(): return Y`
`try`	Try this block, and if exception, go to `except`.	`try: pass`
`while`	While loop.	`while X: pass`
`with`	With an expression as a variable do.	`with X as Y: pass`
`yield`	Pause here and return to caller.	`def X(): yield Y; X().next()`

Data Types:- For data types, write out what makes up each one. For example, with strings, write out how you create a string. For numbers, write out a few numbers.



Type	Description	Example
`True`	True boolean value.	`True or False == True`
`False`	False boolean value.	`False and True == False`
`None`	Represents ”nothing” or ”no value”.	`x = None`
`bytes`	Stores bytes, maybe of text, PNG, file, etc.	`x = b"hello"`
`strings`	Stores textual information.	`x = "hello"`
`numbers`	Stores integers.	`i = 100`
`floats`	Stores decimals.	`i = 10.389`
`lists`	Stores a list of things.	`j = [1,2,3,4]`
`dicts`	Stores a key=value mapping of things.	`e = {'x': 1, 'y': 2}`

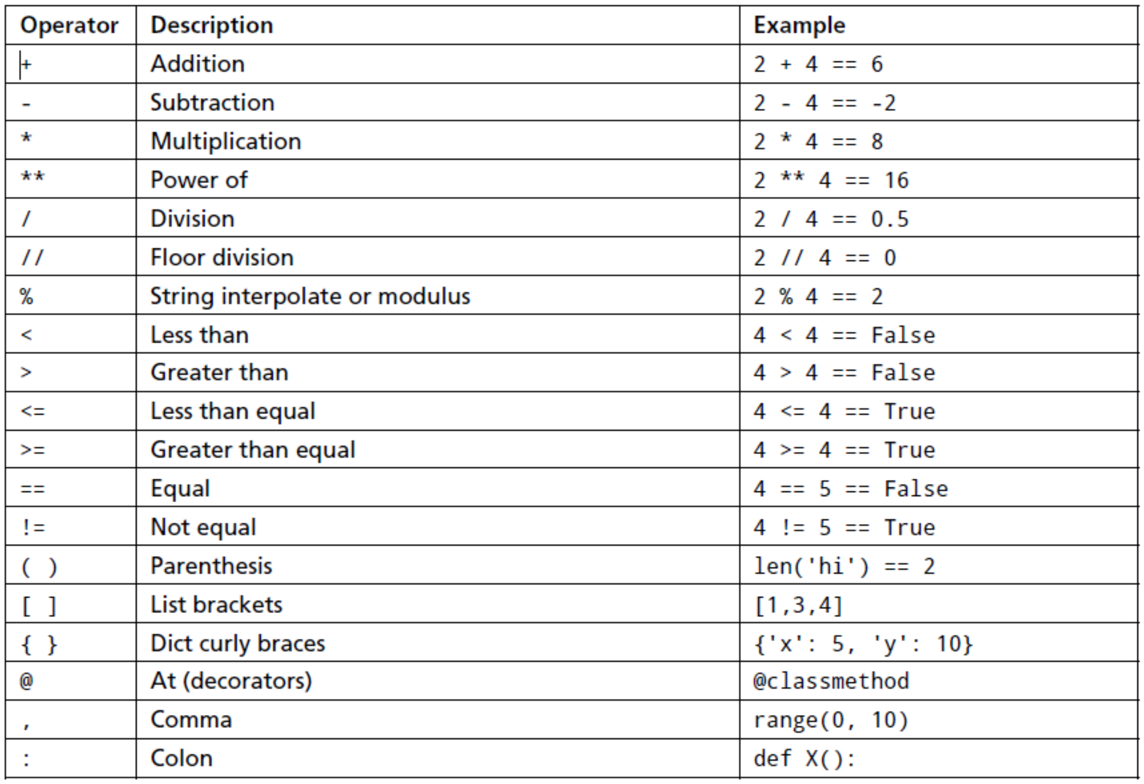
String Escape Sequences:- For string escape sequences, use them in strings to make sure they do what you think they do.

Escape	Description
\\	Backslash
\'	Single-quote
\"	Double-quote
\a	Bell
\b	Backspace
\f	Formfeed
\n	Newline
\r	Carriage
\t	Tab
\v	Vertical tab

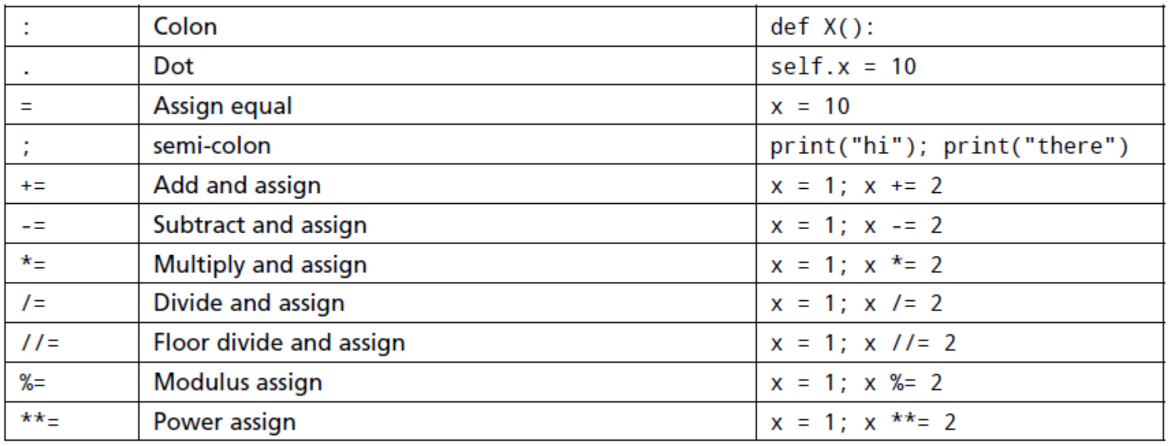
Old Style String Formats (Used in Python2): Same thing for string formats: use them in some strings to know what they do.

Escape	Description	Example
%d	Decimal integers (not floating point).	"%d" % 45 == '45'
%i	Same as %d.	"%i" % 45 == '45'
%o	Octal number.	"%o" % 1000 == '1750'
%u	Unsigned decimal.	"%u" % -1000 == '-1000'
%x	Hexadecimal lowercase.	"%x" % 1000 == '3e8'
%X	Hexadecimal uppercase.	"%X" % 1000 == '3E8'
%e	Exponential notation, lowercase 'e'.	"%e" % 1000 == '1.000000e+03'
%E	Exponential notation, uppercase 'E'.	"%E" % 1000 == '1.000000E+03'
%f	Floating point real number.	"%f" % 10.34 == '10.340000'
%F	Same as %f.	"%F" % 10.34 == '10.340000'
%g	Either %f or %e, whichever is shorter.	"%g" % 10.34 == '10.34'
%G	Same as %g but uppercase.	"%G" % 10.34 == '10.34'
%c	Character format.	"%c" % 34 == '"'
%r	Repr format (debugging format).	"%r" % int == "<type 'int'>"
%s	String format.	"%s there" % 'hi' == 'hi there'
%%	A percent sign.	"%g%%" % 10.34 == '10.34%'

Operators:- Some of these may be unfamiliar to you, but look them up anyway. Find out what they do, and if you still can't figure it out, save it for later



Operator	Description	Example
+	Addition	2 + 4 == 6
-	Subtraction	2 - 4 == -2
*	Multiplication	2 * 4 == 8
**	Power of	2 ** 4 == 16
/	Division	2 / 4 == 0.5
//	Floor division	2 // 4 == 0
%	String interpolate or modulus	2 % 4 == 2
<	Less than	4 < 4 == False
>	Greater than	4 > 4 == False
<=	Less than equal	4 <= 4 == True
>=	Greater than equal	4 >= 4 == True
==	Equal	4 == 5 == False
!=	Not equal	4 != 5 == True
( )	Parenthesis	len('hi') == 2
[ ]	List brackets	[1,3,4]
{ }	Dict curly braces	{'x': 5, 'y': 10}
@	At (decorators)	@classmethod
,	Comma	range(0, 10)
:	Colon	def X():



:	Colon	def X():
.	Dot	self.x = 10
=	Assign equal	x = 10
;	semi-colon	print("hi"); print("there")
+=	Add and assign	x = 1; x += 2
-=	Subtract and assign	x = 1; x -= 2
*=	Multiply and assign	x = 1; x *= 2
/=	Divide and assign	x = 1; x /= 2
//=	Floor divide and assign	x = 1; x //= 2
%=	Modulus assign	x = 1; x %= 2
**=	Power assign	x = 1; x **= 2

How to read a python code:- Go through the code and take note of the following

1. Functions and what they do.
2. Where each variable is first given a value.
3. Any variables with the same names in different parts of the program. These may be trouble later.
4. Any if-statements without else clauses. Are they right?
5. Any while-loops that might not end.
6. Any parts of code that you can’t understand for whatever reason.
7. For every line of code, write what it does; do this for every line of code
8. Lastly, on all of the difficult parts, trace the values of each variable line by line, function by function.

EXERCISE 39: DICTIONARIES

Dictionary is an unordered set of key and value pair. It is a container that contains data, enclosed within curly braces. The pair i.e., key and value is known as item. The key passed in the item must be unique.

The key and the value is separated by a colon(:). This pair is known as item. Items are separated from each other by a comma(,). Different items are enclosed within a curly brace and this forms Dictionary.

get() method with a dictionary returns the value for the given key, if present in the dictionary. If not, then it will return None

Notes about dictionary:-

- Dictionary is mutable i.e., value can be updated.
- Key must be unique and immutable. Value is accessed by key. Value can be updated while key cannot be changed.
- Dictionary is known as Associative array since the Key works as Index and they are decided by the user.

What is the difference between a list and a dictionary?

- A list is for an ordered list of items. A dictionary (or dict) is for matching some items (called ”keys”) to other items (called ”values”).

What would I use a dictionary for?

- When you have to take one value and ”look up” another value. In fact, you could call dictionaries ”look up tables.”

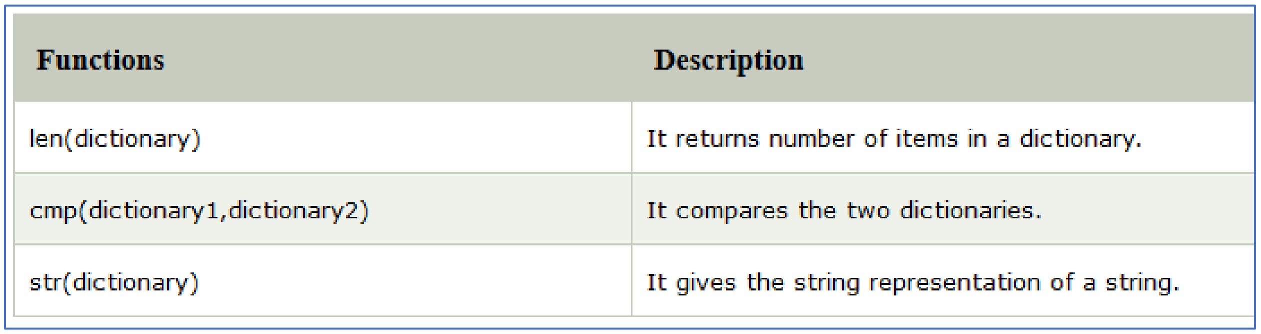
What would I use a list for?

- Use a list for any sequence of things that need to be in order, and you only need to look them up by a numeric index.

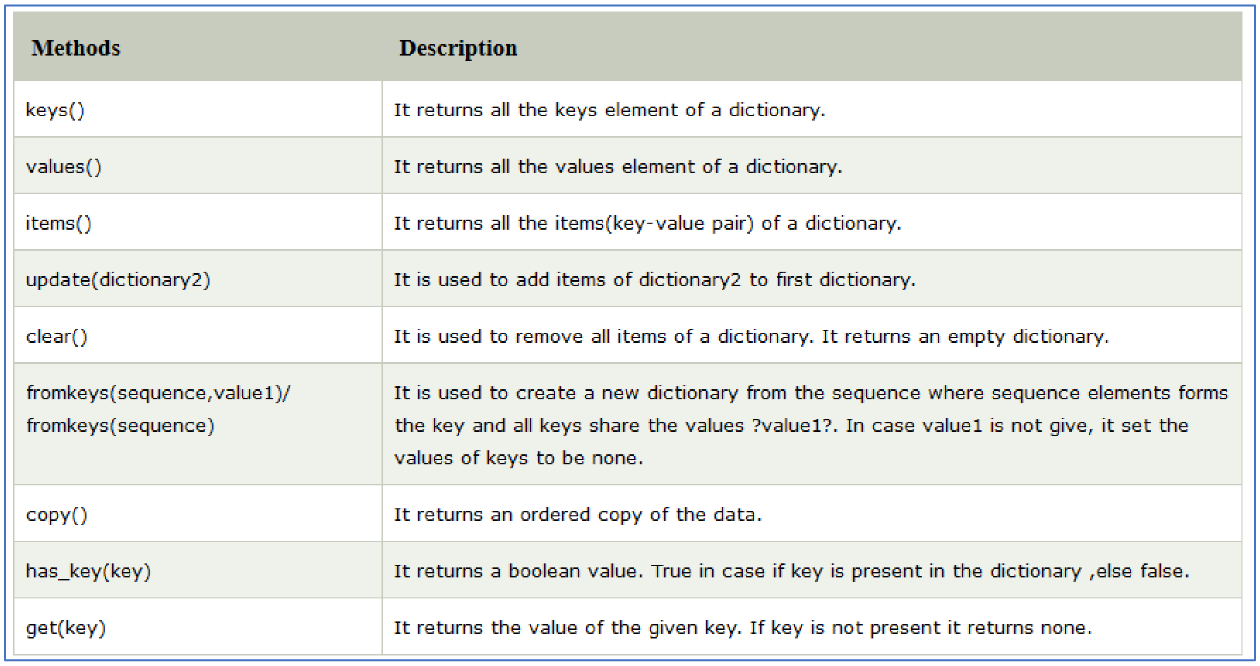
What if I need a dictionary, but I need it to be in order?

- Take a look at the collections.OrderedDict data structure in Python. Search for it online to find the documentation.

Python Dictionary Functions and Methods



Functions	Description
len(dictionary)	It returns number of items in a dictionary.
cmp(dictionary1,dictionary2)	It compares the two dictionaries.
str(dictionary)	It gives the string representation of a string.



Methods	Description
keys()	It returns all the keys element of a dictionary.
values()	It returns all the values element of a dictionary.
items()	It returns all the items(key-value pair) of a dictionary.
update(dictionary2)	It is used to add items of dictionary2 to first dictionary.
clear()	It is used to remove all items of a dictionary. It returns an empty dictionary.
fromkeys(sequence,value1)/ fromkeys(sequence)	It is used to create a new dictionary from the sequence where sequence elements forms the key and all keys share the values ?value1?. In case value1 is not give, it set the values of keys to be none.
copy()	It returns an ordered copy of the data.
has_key(key)	It returns a boolean value. True in case if key is present in the dictionary ,else false.
get(key)	It returns the value of the given key. If key is not present it returns none.

EXERCISE 40: MODULES, CLASSES AND OBJECTS

Keep this idea of ”get X from Y” in your head, and now think about modules. You’ve made a few so far, and you should know they are:

- A Python file with some functions or variables in it ..
- You import that file.
- And you can access the functions or variables in that module with the . (dot) operator.

Here’s how we can access elements from a dictionary and a module

- *mystuff['apple']   # get apple from dict*
- *mystuff.apple()   # get apple from the module*
- *mystuff.tangerine   # same thing, it's just a variable*

You can think about a module as a specialized dictionary that can store Python code so you can access it with the . operator. Python also has another construct that serves a similar purpose called a class. A class is a way to take a grouping of functions and data and place them inside a container so you can access them with the . (dot) operator.

If a class is like a ”mini-module,” then there has to be a concept similar to import but for classes. That concept is called ”instantiate”, which is just a fancy, obnoxious, overly smart way to say ”create.” When you instantiate a class what you get is called an object.

CLASSES AND OBJECTS

- Classes are like blueprints or definitions for creating new mini-modules.
- Instantiation is how you make one of these mini-modules and import it at the same time. ”Instantiate” just means to create an object from the class.
- The resulting created mini-module is called an object, and you then assign it to a variable to work with it.

EXERCISE 41: LEARNING TO SPEAK OBJECT ORIENTED

Word Drills

- **class** Tell Python to make a new type of thing.
- **object** Two meanings: the most basic type of thing, and any instance of some thing.
- **instance** What you get when you tell Python to create a class.
- **def** How you define a function inside a class.
- **self** Inside the functions in a class, self is a variable for the instance/object being accessed.
- **inheritance** The concept that one class can inherit traits from another class, much like you and your parents.
- **composition** The concept that a class can be composed of other classes as parts, much like how a car has wheels.
- **attribute** A property classes have that are from composition and are usually variables.
- **is-a** A phrase to say that something inherits from another, as in a ”salmon” is-a ”fish.”
- **has-a** A phrase to say that something is composed of other things or has a trait, as in ”a salmon has-a mouth.

Phrase Drills

Next I have a list of Python code snippets on the left, and the English sentences for them:

- class X(Y) ”Make a class named X that is-a Y.”
- class X(object): def __init__(self, J) ”class X has-a __init__ that takes self and J parameters.”
- class X(object): def M(self, J) ”class X has-a function named M that takes self and J parameters.”
- foo = X() ”Set foo to an instance of class X.”
- foo.M(J) ”From foo, get the M function, and call it with parameters self, J.”
- foo.K = Q ”From foo, get the K attribute, and set it to Q.”

In each of these, where you see X, Y, M, J, K, Q, and foo, you can treat those like blank spots. For example, I can also write these sentences as follows:

1. ”Make a class named ??? that is-a Y.”

EXERCISE 42: IS-A, HAS-A, OBJECTS & CLASSES

OBJECT ORIENTED PROGRAMMING

Python is an object-oriented programming language. It allows us to develop applications using Object Oriented approach. In Python, we can easily create and use classes and objects.

Major principles of object-oriented programming system are given below

- Object
- Class
- Method
- Inheritance
- Polymorphism
- Data Abstraction
- Encapsulation

Object:- Object is an entity that has state and behavior. It may be anything. It may be physical and logical. For example: mouse, keyboard, chair, table, pen etc.

Everything in Python is an object, and almost everything has attributes and methods. All functions have a built-in attribute __doc__, which returns the doc string defined in the function source code.

Class:- Class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods. For example: if you have an employee class then it should contain an attribute and method i.e. an email id, name, age, salary etc. So while employee is a class we can create many objects from this class like name, age etc.

There are also some special attributes that begins with double underscore (__). For example: __doc__ attribute. It is used to fetch the docstring of that class.

Method:- Method is a function that is associated with an object. In Python, method is not unique to class instances. Any object type can have methods.

Inheritance:- Inheritance is a feature of object-oriented programming. It specifies that one object acquires all the properties and behaviors of parent object. By using inheritance you can define a new class with a little or no changes to the existing class. The new class is known as derived class or child class and from which it inherits the properties is called base class or parent class. It provides re-usability of the code.

Polymorphism:- Polymorphism is made by two words "poly" and "morphs". Poly means many and Morphs means form, shape. It defines that one task can be performed in different ways.

Encapsulation:- Encapsulation is also the feature of object-oriented programming. It is used to restrict access to methods and variables. In encapsulation, code and data are wrapped together within a single unit from being modified by accident.

Data Abstraction:- Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonym because data abstraction is achieved through encapsulation.

Abstraction is used to hide internal details and show only functionalities. Abstracting something means to give names to things, so that the name captures the core of what a function or a whole program does.

Some concepts in OOB:

- **Abstraction**:- humans use it most of the time, for example people drive car but don't know how all internals are working they just need to know how to use accelator, brake etc that is people use the them but don't need know how it is working. Similarly, you can use method of object without knowing how it is doing, all its function. However most of the time you going to write your own methods.
- **Encapsulation**:- Let us suppose a Television. If this was sold without the plastic cover, then any one passing by it will change something in it and may lead to failure. So the company producing covers it up and provides limited freedom to users. This is encapsulation : Hiding the internals of a class
- **Polymorphism**:- In a house we have three servants for doing three different jobs, lets say Servant 1 for buying milk, Servant 2 for buying vegetable & Servant 3 for buying grocery. The problem is for assigning the task we have to remember the names of all three servants and also remember that who is responsible for which task. The solution to this is We will assign all three tsk to a single person and he will do so as per our orders here we are overloading the task of purchasing to one person
- **Inheritance**:- First visiblity mode Public: For eg my father has written in his will that his car will be inherited by me for PUBLIC use then I and my family members can use it. If it has been written that car will be inherithed by me for my PRIVATE use so only I can use it no outer person.