**MY PYTHON LEARNING**

**UPDATED**: 12/05/2019

**INDEX**

[**GETTING STARTED**](#GETTING_STARTED)

**[BEST PRACTICES](#BEST_PRACTICES)**

[**BEST FUNCTIONS**](#BEST_FUNCTIONS)

[**BEST CLASSES**](#BEST_CLASSES)

[**COMMAND PROMPT FUNCTIONS**](#COMMAND_PROMPT)

[**PYTHON FUNCTIONS**](#PYTHON_FUNCTIONS)

* [**MODULE 1**](#MODULE_1)
* [**MODULE 2**](#MODULE_2)
* [**MODULE 3**](#MODULE_3)
* [**MODULE 4**](#MODULE_4)
* [**MODULE 5**](#MODULE_5)
* [**MODULE 6**](#MODULE_6)
* [**MODULE 7**](#MODULE_7)
* [**MODULE 8**](#MODULE_8)
* [**MODULE 9**](#MODULE_9)

**[GLOSSARY](#GLOSSARY)**

[**REFERENCES**](#REFERENCES)

**QUESTIONS**

1. Using While, when would you need to use “else” and when do you not because it works ways.

2. The parts in the Module 3 YouTube video regarding True / False because I’m not sure how it’s used in practice.

<https://youtu.be/EoCXrbkdI0c?t=2194>

**Current YouTube Video**

<https://youtu.be/EoCXrbkdI0c?t=3562>

[**GETTING STARTED**](#_top)

**IDE**

There are many Python programs called integrated development environment (IDE). Popular ones are **Spyder** (used by Boeing), **PyCharm** (used by UW Class) and **IDLE** (used by Raspberry Pi).

# Interactive Mode – Pressing enter after a function will run and display immediate results.

# Script Mode – Like a text editor; pressing enter will not run results.

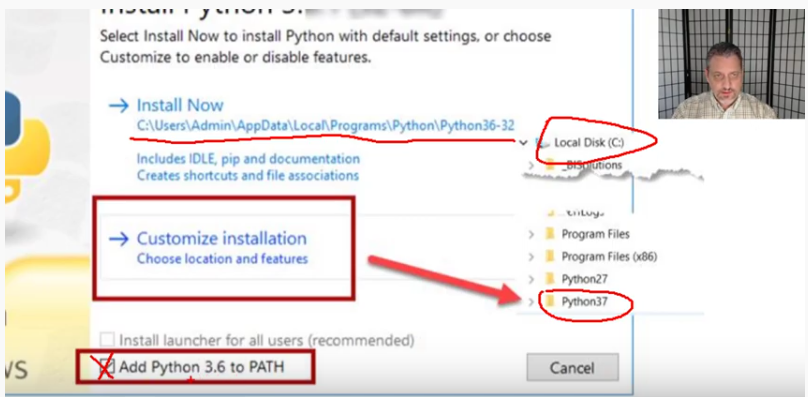
# POPULAR TEXT EDITORS

# Sublime Text

# Atom

**Download Pythons (comes with IDLE IDE):**

https://www.python.org/downloads



**Determine path to search for executable file:**

C:\Users\oe172e>**path**

**Display all the python features**

C:\Users\oe172e>**python -?**

**Display network connectins**

C:\Users\oe172e>ipconfig

**Run python program from command prompt**

C:\Users\oe172e>python "C:\filepath\game\_over.py"

* Go to the file and copy the path and paste in front of “python” in the above line.

**You can use any text editor, example notepad, to write the file but just save as filename.py**

**Python is case-sensitive. Therefore the variable x and X are two different variables!**

**[BEST PRACTICES](#_top)**

**1. CONSTANTS are always in UPPERCASE**

**2. variables are in lowercase**

**3. Main function is typically called main()**

**[BEST FUNCTIONS](#_top)**

**1. Ask a yes or no question and return a response**

def ask(blah):

response = None

while response not in ("y", "n"):

response = input(blah).lower()

return response

answer = ask("\nPlease enter 'y' or 'n': ")

print("Thanks for entering:", answer)

**2. To assign a global variable from the local variable in a function**

def test():

a = 5 # define local variable “a”

b = 8 # define local variable “b”

return a, b # “a” and “b” and shot out from function test()

c, d = test() # note I don’t use the same names for global vs local variables

print(c)

print(d)

**Output**

5

8

**3. To create a list with blank characters for each item**

MAX = 3 # this value determines how many characters in the list

EMPTY = " " # this value deteremines what that character will be

def new\_board():

board = [] # defines board as a list

for i in range(MAX): # iterates from 0 to MAX

board.append(EMPTY) # adds new item to the list called “board”

return board # shoots out the list

print(new\_board())

**Output**

[' ', ' ', ' ']

**4. To print the list from Function new\_board above**

NUM\_SQUARES = 9

EMPTY = " "

def new\_board():

board = []

for square in range(NUM\_SQUARES):

board.append(EMPTY)

return board

def display\_board(board):

print("\n\t", board[0], "|", board[1], "|", board[2])

board = new\_board()

display\_board(board)

**Output**

**| |** # note that there are 3 space characters displayed

**5. How to remove an item from a list by iteration**

my\_list = [**"apple"**, **"banana"**, **"peach"**]  
print(**f'Initial list is: {**my\_list**}'**)  
ans = input(**"Which fruit would you like to remove: "**)  
b = **False** *# Create a boolean Flag in case user inputs a item NOT in the list*i = 0

**while** i < len(my\_list):  
 **if** ans == my\_list[i]:  
 **del** my\_list[i]  
 b = **True** i += 1 *# iterates i to go up the list and determine if this is the item user chooses to delete*

**if** b == **True**:  
 print(**f'Updated list is now: {**my\_list**}'**)  
**else**:  
 print(ans + **" is not in the list"**)

**6. To print row of items in a list as a column of items**

my\_list = [**"apple"**, **"banana"**, **"peach"**]  
print(my\_list)  
**for** i **in** my\_list:  
 print(i)

**7. To create a text file automatically if one does not exist currently is to add this two line of code:**

**objFile = open(file\_name, "a")**  
**objFile.close()**

**Note that you still need to write the rest of the code to read from the text file:**

**objFile = open(file\_name, "a")**  
**objFile.close()**

       file = open(file\_name, "r")  
       for line in file:  
        data = line.split(",")  
             row = {"Task": data[0].strip(), "Priority": data[1].strip()}  
             list\_of\_rows.append(row)  
       file.close()  
       return list\_of\_rows

**8. Save Data to Text File**

*# Data ---------------------------------­­­------------- #*

strData = 'blah blah' # global variable

strFileName = 'blahfile.txt' # global variable

*# Processing --------------------------------------- #*

def save\_data(data, file\_name): # data and file\_name are local variables

*""" Saves string data to a file*

:param *data: (string) with data to save*

:param *file\_name: (string) with name of file*

:return*: nothing*

*"""*

file = open(file\_name, "w") # w means write

file.write(data + "\n")

file.close()

*# Presentation ------------------------------------- #*

save\_data(strData, strFileName)

# the position of the arguments “strData” and “strFileName” is important.

print('Data Saved!')

**9. Read Data from Text File**

*# Data --------------------------------------------- #*

strFileName = **blahfile.txt'**

*# Processing --------------------------------------- #*

**def** read\_data(file\_name):

*""" Reads all string data from a file*

**:param** *file\_name: (string) with name of file*

**:return***: (string) of data read from the file*

*"""*

file = open(file\_name, **"r"**)

data = file.read() *# read all the data in the file at once*

file.close()

**return** data

*# Presentation ------------------------------------- #*

print(**'Here is the data from the file:'**)

print(read\_data(file\_name=strFileName))

print(**'^ Note the extra blank line was read from the file too! ^'**)

**10. Read Data from Text File and Append to Dictionary List (Using Class)**

strFileName = "testtest.txt"

lstTable = []

class FileProcessor:

@staticmethod

def ReadFileDataToList(file\_name):

"""

Desc - Reads data from a file into a list of dictionary rows

:param file\_name: (string) with name of file:

:return: (list) of dictionary rows

"""

list\_of\_dictionary\_rows = []

file = open(file\_name, "r")

for line in file:

data = line.split(",")

row = {"Task": data[0].strip(), "Priority": data[1].strip()}

list\_of\_dictionary\_rows.append(row)

file.close()

return list\_of\_dictionary\_rows

def main():

lstTable = FileProcessor.ReadFileDataToList(strFileName)

print(lstTable)

main()

**10. Add New Data to Text File**

# Data --------------------------------------------- #

strFileName = 'blahfile.txt'

data = 'extra blah blah'

# Processing --------------------------------------- #

def add\_more\_data(data, file\_name):

""" Saves string data to a file using APPEND Mode

:param data: (string) with data to save

:param file\_name: (string) with name of file

:return: nothing

"""

file = open(file\_name, "a")

file.write("\n" + data + "\n")

file.close()

# Presentation ------------------------------------- #

add\_more\_data(data, file\_name=strFileName)

print("Data Added!")

**11. Read One Line of Data from Text File using readline()**

# Data --------------------------------------------- #

strFileName = 'blahfile.txt'

# Processing --------------------------------------- #

def read\_data\_row(file\_name):

"""

Reads a row of string data from a file

:param file\_name: (string) with name of file

:return: (string) with one row of data from the file

"""

file = open(file\_name, "r")

# readline() acts like a "cursor"

data = file.**readline()** # read one row of data in the file

file.close()

return data

# Presentation ------------------------------------- #

print('Here is the first row of data from the file:')

print(read\_data\_row(file\_name=strFileName))

print('Here is the SAME row of data from the file:')

print(read\_data\_row(file\_name=strFileName))

**Output**

Here is the first row of data from the file:

blah 1

Here is the SAME row of data from the file:

blah 1

**12. Loop through Read One Line of Data from Text File using readline()**

# Data --------------------------------------------- #

strFileName = 'blahfile.txt'

# Processing --------------------------------------- #

def read\_some\_data\_rows(file\_name, number\_of\_rows):

""" Reads rows of string data from a file

:param file\_name: (string) with name of file

:param number\_of\_rows: (int) with number of rows you want from the file

:return: (list) with one or more data rows read from the file

"""

counter = 0

data = []

file = open(file\_name, "r")

while counter < number\_of\_rows:

data.append([file.readline()]) # APPENDING the data to a list

counter += 1

file.close()

return data # returning the chosen row of data

# Presentation ------------------------------------- #

print('Here is the first row of data from the file:')

print(read\_some\_data\_rows(file\_name=strFileName, number\_of\_rows=1))

# new line character is not included!

print('Here is the first AND second row of data from the file:')

print(read\_some\_data\_rows(file\_name=strFileName, number\_of\_rows=2))

**Output**

Here is the first row of data from the file:

[['blah 1\n']]

Here is the first AND second row of data from the file:

[['blah 1\n'], ['blah 2\n']]

**13. Read Specific Row of List from a Text File**

# Data --------------------------------------------- #

strFileName = 'blahfile.txt'

# Processing --------------------------------------- #

def read\_a\_data\_row(file\_name, row\_you\_want):

"""

Reads rows of string data from a file

:param file\_name: (string) with the name of file

:param row\_you\_want: (int) with the number of the row you want from the file

:return: (string) with one or more data rows read from the file

"""

counter = 0

file = open(file\_name, "r")

while counter < row\_you\_want:

data = [file.readline()] # REPLACING the data in a list

counter += 1

file.close()

return data # returning the chosen row of data

# Presentation ------------------------------------- #

print('Here is the second row of data from the file:')

print(read\_a\_data\_row(file\_name=strFileName, row\_you\_want=2))

print('Here is the third row of data from the file:')

print(read\_a\_data\_row(file\_name=strFileName, row\_you\_want=3))

**Output**

Here is the second row of data from the file:

['blah 2\n']

Here is the third row of data from the file:

['blah 3\n']

**14. Read Specific Row of Data From Text File**

# Data --------------------------------------------- #

strFileName = 'blahfile.txt'

# Processing --------------------------------------- #

def read\_file\_data\_to\_list(file\_name):

"""

Reads rows of data data from a file into a list

:param file\_name: (string) with name of file

:return: (list) of data rows read from the file

"""

data = [] # you must initialize the list variable before you use it

for row in open(file\_name, 'r'):

data.append(row) # read one row of data in the file per loop

# automatically closes the file

return data

# Presentation ------------------------------------- #

print('Here are the two rows of data from the file:')

print(read\_file\_data\_to\_list(file\_name=strFileName)[0].strip())

print(read\_file\_data\_to\_list(file\_name=strFileName)[2].strip())

**Output**

Here are the two rows of data from the file:

blah 0

blah 2

**15. Pickle and Unpickle One Line of Data from Binary File**

import pickle # This imports code from another code file!

# Data -------------------------------------------- #

strfilename = 'AppData.dat'

lstCustomer = []

# Processing -------------------------------------- #

def save\_data\_to\_file(file\_name, list\_of\_data):

# Now we store the data with the pickle.dump method

objFile = open(file\_name, "ab") # a = append, b = binary file

pickle.dump(list\_of\_data, objFile)

objFile.close()

def read\_data\_from\_file(file\_name):

# And, we read the data back with the pickle.load method

objFile = open(file\_name, "rb")

list\_of\_data = pickle.load(objFile) #load() only loads one row of data

objFile.close()

return list\_of\_data

# Presentation ------------------------------------ #

# Get ID and NAME From user, then store it in a list object

intId = int(input("Enter an Id: "))

strName = str(input("Enter a Name: "))

lstCustomer = [intId, strName]

# Store the list object into a binary file

save\_data\_to\_file(strfilename,lstCustomer)

# Read the data from the file into a new list object and display the contents

print(read\_data\_from\_file(strfilename))

**Output**

Enter an Id: 1

Enter a Name: Apples

[1, 'Apples']

**BEST CLASSES**

1. **Write from Dictionary to a File**

strFileName = 'products.txt' # text file

lstTable = [{'product\_name': 'Hello', 'product\_price': '$1'}, {'product\_name': 'World', 'product\_price': '$2'}] # list of dictionary

class FileProcessor:

@staticmethod

def write\_file\_from\_list\_of\_dictionaries(file\_name, list\_of\_dictionary\_rows):

""" Write data to a file from a list of dictionary rows

:param file\_name: (string) with name of file

:param list\_of\_dictionary\_rows: (list) of dictionary data saved to file

:return: (bool) with status of success status

"""

success\_status = False

file = open(file\_name, "w")

for row in list\_of\_dictionary\_rows:

file.write(row["product\_name"] + "," + row["product\_price"] + "\n")

file.close()

success\_status = True

return success\_status

def main():

FileProcessor.write\_file\_from\_list\_of\_dictionaries(strFileName, lstTable)

main()

1. **Initialise an object class**

class Critter:

def \_\_init\_\_(self, my\_name, my\_color, my\_money):

print("A new critter has been born!”)

self.name = my\_name

self.color = my\_color

self.money = my\_money

def talk(self):

print("I'm", self.name)

print("My color is", self.color)

print("My pocket has $", self.money)

print()

def main():

crit1 = Critter(my\_name = "Apple", my\_color = "Brown", my\_money = 1.56)

crit1.talk()

crit2 = Critter(my\_name = "Banana", my\_color = "Yellow", my\_money = 5.99)

crit2.talk()

main()

**Output**

A new critter has been born!

I'm Apple

My color is Brown

My pocket has $ 1.56

A new critter has been born!

I'm Banana

My color is Yellow

y pocket has $ 5.99

[**COMMAND PROMPT OR MS-DOS FUNCTIONS**](#_top)

**Open Command Prompt (CMD)**

**Type in “Python –V” to determine which version of python is installed.**

**Create a Batch (bat) File**

**Batch files is an executable file in MS Windows you can use to open applications like python scripts.**

**In Notepad, type in the following text and save the file with .bat extension:** “C:\Users\oe172e\AppData\Local\Continuum\anaconda3\python.exe C:\\_PythonClass\Assignment01-COMPLETED\test.py”

**Note: The first sentence is the path to the python program and the send sentence is the path to the python script.**

**pause**

#Can be used in a batch file too; allows the user to press a key before program ends.

**Determine path to python.exe**

In command prompt, key in “python” first, then press enter. Then…

>>> **import os**

>>> **import sys**

>>> **os.path.dirname(sys.executable)**

'C:\\Users\\oe172e\\AppData\\Local\\Continuum\\anaconda3'

**cd C:\\_PythonClass**

#change directory

**del file\_name**

#deletes a file

**dir**

#list the contents in the current folder or directory

**mkdir folder\_name**

#create a new folder or directory in the current folder named “folder\_name”

**rmdir folder\_name**

#deletes a folder named “folder\_name”

**move c:\windows\temp\\*.\* c:\temp**

#move file form one folder to another

**move c:\Users\oe172e\Apple\Blah.txt c:\Users\oe172e\Banana**

#To move a file Blah.txt from a folder Apple to folder Banana using Command Prompt

**exit**

#exit the command prompt window

**ren "current\_filename.ext" "new\_filename.ext"**

**#rename a file**

**type nul > file\_name.txt**

#create a new text file call “file\_name” in the current directory.

**[PYTHON FUNCTIONS](#_top)**

**[MODULE 1](#_top)**

**[MODULE 2](#_top)**

**number=100**

**while (number >= 0):**

**print (number)**

**number = number – 2**

#Countdown from 100 to 0, in steps of 2 and prints the numbers.

**print(“text”)**

**print(‘text’)**

# prints whatever text in the quotation marks. Option to use “ or ‘. Makes no difference.

**print(variable)**

**input("\n\nPress the enter key to exit.")**

The Enter key will close the program window.

**Input()**

#Does the same thing as above where it pauses and waits for a key to be pressed before ending the program. However, it doesn’t print the sentence “Press the enter key to exit.”

**print("Here", end=" ")**

# prints “Here” but does not start a new print function from a newline but after the word “Here”.

**print( """**

**< >**

**+**

**----------**

**""")**

# Any item within the triple quotation marks will be printed as is.

**print("\t\t\tFancy Credits")**

# “\t” is used at a Tab.

**print(“\nSpecial thanks goes out to:”)**

#The “\n” used in a sequence like that adds a new line before “Special…”

#it works the same as using “print()”.

**print("\a")**

#Sound the system bell; chime.

**print("Pie" \* 10)**

#prints “PiePiePiePiePiePiePiePiePiePie”

**7 / / 3**

#prints “2”. It prints the division results as an interger, meaning no decimal places.

**107 % 4**

#prints the remainder only.

**name = input("Hi. What's your name? ")**

#assignes variable “name” and look for input from keyboard.

**quote = "I think there is a world market for maybe five computers."**

**print(quote.upper())**

#prints “I THINK THERE IS A WORLD MARKET FOR MAYBE FIVE COMPUTERS.”

**quote = "I think there is a world market for maybe five computers."**

**print(quote.lower())**

#prints “i think there is a world market for maybe five computers.”

**quote = "I think there is a world market for maybe five computers."**

**print(quote.title())**

#prints “I Think There Is A World Market For Maybe Five Computers.”

**quote = "I think there is a world market for maybe five computers."**

**print(quote.replace("five", "millions of"))**

#prints “I think there is a world market for maybe millions of computers.”

**quote = "I think there is a world market for maybe five computers."**

**print(quote)**

#print “I think there is a world market for maybe five computers.”

**car = input("Enter Car Cost: ")**

**car = int(car)**

**OR**

**car = int(input("Enter Car Cost: "))**

#print “Enter Car Cost: “ and waits for an keyboard input to assign to interger “car”.

#both ways work but the second way is a lot more efficient.

**‘’’**

**print("test2")**

**‘’’**

**OR**

**“””**

**print("test2")**

**“””**

Any code inside the triple quoatations ‘’’ ‘’’ OR “”” “”” will be marked as comments. This is good to block out a code that you think is problematic to see if the error goes away.

**exit()**

#exit the python interactive interface and go back to command prompt.

**Answer=Input()** #defines variable Answer first and seek input

**if(Answer == “A”):**

**print (“True”)**

**else:**

**print(“False”)**

#if, else function

**def DemoMethod()**

**code, code, code**

**#end DemoMethod**

#def creates a function “DemoMethod”. A function a group of organized, usable code. You run the function by calling it out in the code, just like print(), you call out DemoMethod(). The note “#end DemoMethod” is not neccessay but highly recommended.

**Calculating Power of**

Two calculate 3^2: type in **3 \*\* 2**

# String format() Method

Links:

<https://www.w3schools.com/python/ref_string_format.asp>

**Example 1 - Insert the price inside the placeholder, the price should be in fixed point, two-decimal format:**

txt = "For only {price:.2f} dollars!"

print(txt.format(price = 49))

**Output**

For only 49.00 dollars!

**Example 2**

# named indexes:

txt1 = "My name is {fname}, I'm {age}".format(fname = "John", age = 36)

# numbered indexes:

txt2 = "My name is {0}, I'm {1}".format("John",36)

# empty placeholders:

txt3 = "My name is {}, I'm {}".format("John",36)

print(txt1)

print(txt2)

print(txt3)

**Output**

My name is John, I'm 36  
My name is John, I'm 36  
My name is John, I'm 36

**Concatenating Strings**

**#Code**

output = "hello" + " " + "world"

print(output)

**#Output**

hello world

**#Code**

output = "hello" + "world"

print(output)

**#Output**

helloworld

**#Code**

output = "hello" + " " + " " + "world"

print(output)

**#Output**

hello world

**x\_list = [apple, banana, cantalop]**

#creates a list of items x, y, z and assign to variable x\_list

x\_list[0] = apple

x\_list[1] = banana

x\_list[2] = cantalop

**Example:**

print(x\_list[2]) will output **cantalop**

print (x\_list[0:2]) will output **apple banana**

**list[]**

**2 dimensional list**

# a list will also work on a word like “cat” where “c”, “a”, and “t” are individual items in the list or string

**Example**

word = "cat"

print(word[1])

print(word[0]) # note that the first item in the list is “c” and represented by 0

**Output**

a

**c**

**list.append()**

**# adds to a list**

**Example**

fruit = ['apple','banana'] # list is ['apple', 'banana']

fruit.append('orange') # list is now ['apple', 'banana', 'orange']

**list.extend()**

**# adds individual characters to the list**

**Example**

fruit = ['apple','banana'] # list is ['apple', 'banana']

fruit.extend('orange') # list is now ['apple', 'banana', 'o', 'r', 'a', 'n', 'g', 'e']

**Slicing using list[]**

# to use or print part of list

**Example 1 – use the first parts of the list before and including the item 4**

# remember that item 4 is “e” from below because we count from 0

message = "abcde"

print(message[:4])

**Output:**

abcd

**Example 2 – use the last parts of the list after and including item 3**

# remember that item 3 is “4” below because we count from 0

message = “abcde”

print(message[3:])

**Output:**

de

**upper()**

**print(message.upper())**

**Output:**

HELLO WORLD

**lower()**

**print(mesage.lower())**

**Output:**

hello world

**count()**

**print(message.count(‘Hello’))**

**Output:**

1

#counts the number of time the word “Hell” appears in “Hello World”

**print(message.count(‘l’))**

**Output:**

3

#counts the number of time the letter “l” appears in “Hello World”

**find()**

**print(message.find(‘World’))**

**Output:**

6

#Finds the word “World” and show that it starts in the 6th place of “Hello World”

**replace()**

message = message.replace(‘World’, ‘Universe’)

print(message)

**Output:**

Hello Universe

#note that you have to reset the variable message when you use replace(). If you don’t want to do that, you can also the following:

new\_message = message.replace(‘World’, ‘Universe’)

print(new\_message)

**Output:**

Hello Universe

**x = object()**

#defines x as a variable

Example: See <https://www.learnpython.org/en/Basic_Operators>

**len()**

**print(len(variable))**

#The len() function returns the number of items of an object. len is short for length.

Example: see <https://www.learnpython.org/en/Basic_Operators>

**Example:**

print(len(“table”))

**Output:**

5 # the word “table” has a length of 5 alphabets

**Datetime**

import datetime

print(datetime.datetime.now())

print(datetime.datetime.now().strftime('%A'))

date = datetime.datetime.now().strftime('%A')

print(date)

**Output:**

2019-10-10 07:12:45.839466

Thursday

Thursday

#The 1st line output is today’s date, inclusing microseconds at the end.

#2nd line output includes a method called strftime(), and takes one parameter, format, to specify the format of the returned string.

#link: <https://www.w3schools.com/python/python_datetime.asp>

**A reference of all the legal format codes**

|  |  |  |
| --- | --- | --- |
| **Directive** | **Description** | **Example** |
| %a | Weekday, short version | Wed |
| %A | Weekday, full version | Wednesday |
| %w | Weekday as a number 0-6, 0 is Sunday | 3 |
| %d | Day of month 01-31 | 31 |
| %b | Month name, short version | Dec |
| %B | Month name, full version | December |
| %m | Month as a number 01-12 | 12 |
| %y | Year, short version, without century | 18 |
| %Y | Year, full version | 2018 |
| %H | Hour 00-23 | 17 |
| %I | Hour 00-12 | 5 |
| %p | AM/PM | PM |
| %M | Minute 00-59 | 41 |
| %S | Second 00-59 | 8 |
| %f | Microsecond 000000-999999 | 548513 |
| %z | UTC offset | 100 |
| %Z | Timezone | CST |
| %j | Day number of year 001-366 | 365 |
| %U | Week number of year, Sunday as the first day of week, 00-53 | 52 |
| %W | Week number of year, Monday as the first day of week, 00-53 | 52 |
| %c | Local version of date and time | Mon Dec 31 17:41:00 2018 |
| %x | Local version of date | 12/31/2018 |
| %X | Local version of time | 17:41:00 |
| %% | A % character | % |

**Strings – Working with Contextual Data**

#There are 3 ways to code this with the same output

#The 1st way has limitations that you cannot combined strings (or words) and numbers

#The 3rd and latest way is call f-string or f-strings

#The nice thing about the f-strings is that you can add methods like the upper()

#Note the 5th Way works too without creating a new variable called “greeting”

greeting = 'Hello'  
name = 'Kevin'

#1st Way  
message = greeting + ' ' + name + "!"  
print(message)

#2nd Way  
message = '{} {}!'. format(greeting, name)  
print(message)

#3rd Way  
message = f'{greeting} {name}!'  
print(message)

4th Way

message = f'{greeting} {name.upper()}!'  
print(message)

5th Way

print(f’{greeting} {name}’)

**Output:**

Hello Kevin!  
Hello Kevin!  
Hello Kevin!

Hello KEVIN!

Hello Kevin!

**List out Valid Attributes**

name = 'Kevin'

print(dir(name))

Output:

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_

**type()**

interger\_variable=5

float\_variable=2.0

string\_variable = "happy"

boolean\_variable = True

print(type(interger\_variable))

print(type(float\_variable))

print(type(string\_variable))

print(type(boolean\_variable))

**Output:**

<class 'int'>

<class 'float'>

<class 'str'>

<class 'bool'>

| **Escape Sequence** | **Meaning** |
| --- | --- |
| \*newline* | Ignored; recognizes words after the backslash as a continuous sentence. |
| \\ | Backslash (\) |
| \' | Single quote (') |
| \" | Double quote (") |
| \a | ASCII Bell (BEL) |
| \b | ASCII Backspace (BS) |
| \f | ASCII Formfeed (FF) |
| \n | ASCII Linefeed (LF) |
| \r | ASCII Carriage Return (CR) |
| \t | ASCII Horizontal Tab (TAB) |
| \v | ASCII Vertical Tab (VT) |
| \*ooo* | ASCII character with octal value *ooo* |
| \x*hh...* | ASCII character with hex value *hh...* |

# sep and end parameter in print()

# 

# print("high", "low", sep="-")

# print("high", "low", sep="@")

# print("high", "low", sep="---lala---", end=" baba")

# Output:

# high-low

# high@low

high---lala---low baba

**list()**

**dict()**

**tuple()**

**set()**

Python data structures include lists, dictionaries, tuples, and sets.

List, dictionary and set items are mutable. Tuple items are immutable.

Lists and tuples maintain order. Dictionaries and sets are unordered.

Lists and tuples allow duplication. Dictionary and set items are unique.

List, dictionary, tuple, and set items may be accessed using a for loop.

List and tuple items may be accessed by index. Dictionary items are accessed by key. Set items cannot be accessed by index.

See <https://en.wikiversity.org/wiki/Python_Programming/Tuples_and_Sets>

**float(x)**

a = 10

print(a)

**Output:**

10

a = 10

print(float(a))

**Output:**

10.0

**To change the number of decimal places**

a = 10

print({:.3f}.format(a))

**Output**

10.000

**tuple()**

this\_tuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

print(thistuple) # prints ('apple', 'banana', 'cherry', 'orange', 'kiwi', 'mango')

print(thistuple[1]) # prints banana

print(thistuple[-1]) # prints mango

print(thistuple[2:5]) # prints ('cherry', 'orange', 'kiwi')

print(thistuple[-4:-1]) # prints ('cherry', 'orange', 'kiwi')

print()

#Iterate through the items and print the values:

for x in this\_tuple:

print(x)

print()

#Check if "apple" is present in the tuple:

if "apple" in thistuple:

print("Yes, 'apple' is in the fruits tuple")

print()

#Use the add method to add "orange" to the tuple set

thistuple.add("orange")

#Print the number of items in the tuple:

print(len(thistuple))

#The del keyword can delete the tuple completely

del thistuple

#To join two or more tuples you can use the + operator:

tuple1 = ("a", "b" , "c")

tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2

print(tuple3)

**Output:**

('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango')

banana

mango

('cherry', 'orange', 'kiwi')

('orange', 'kiwi', 'melon')

apple

banana

cherry

orange

kiwi

melon

mango

Yes, 'apple' is in the fruits tuple

7

('a', 'b', 'c', 1, 2, 3)

**not**

#Use this “not” sparingly because it is more difficult to read. Use expression “!=” instead.

x = 1

if (**not**(x == 1):

print(“It is true”)

else:

print(“It is false)

**!=**

**#This is the expression Not Equal To**

x = 1

if (x **!=** 1)

print(“It is true”)

else:

print(“It is false)

**[MODULE 3](#_top)**

**random**

The random module gives access to various useful functions and one of them being able to generate random numbers, which is randint().

**random.randint()**

An inbuilt function of the random module in Python3

import random

die = random.randint(1, 6) #randomly choses a interger between 1 and 6

print(f'You roll a {die}')

**Output:**

You roll a 3

**random.choice()**

# pick one word randomly from the sequence

**Example:**

import random

WORD\_LIST = ("table", "chair", "cabinet")

# pick a random choice from WORD\_LIST and assigns to variable word\_pick

word\_pick = random.choice(WORD\_LIST)

print(word\_pick)

**random.randrange()**

**# pick a random number within the range**

**Example**

import random

word\_pick = "television"

position = random.randrange(len(word\_pick))

print(position)

**Output:**

5 # prints a random number between 1-10 because “television” has 10 alphabets

**if**

**else:**

password = input("Enter your password: ")

if password == "secret":

print("Access Granted")

else:

print(“Access Denied”)

**Output:**

Enter your password: secret

Access Granted

**elif**

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition". You can use as many elif as you like.

import random

mood = random.randint(1, 3)

if mood == 1:

print(“happy”)

elif mood == 2:

print(“neutral”)

elif mood == 3:

print(“sad”)

else:

print("Illegal mood value!”)

**while**

**else:**

With the while loop we can execute a set of statements as long as a condition is true. Be careful not to create an inifinite loop. You can include more than one condition by using the operators "**or"** and **"and."** If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

**Example 1**

response = "" #this is called a sentry variable. It is important to assign a empty variable.

while response != "Because":

response = input("Why?\n")

print("Oh. Okay.")

**Example 2**

while apple > 0 and banana > 0:

print(“Happy”)

else:

print(“Bad”)

**while True:**

**break**

**continue**

**Example 1 # Script will loop unti an input of “0”**

var = ""

**while** **True**:

var = input("Type in a string to echo or Enter 0 to quit!")

if(var == "0"): **break # stops the loop when user inputs “0”**

if(var == “c”): **continue # input “c” will skip the rest of the loop and go back to the start of the loop**

else: print(var) # Feels redundant to use “else”, but you need it if you want to use the break fuction to skip what’s in the else:

**Example 2**

count = 0

while True:

count += 1

if count > 10: # end loop if count greater than 10

break

if count == 5: # skip all code below “continue”.

continue

print(count)

**Output:**

1

2

4

5

# Note that 3 is missing becaused it is skipped because we use “continue”.

**Example 3** # this example shows the abosolute need to use the “else”.

secretNum = 8

userGuess = int(input("I'm thinking of a number 1 to 10. Your guess? >>"))

while(True):

if(userGuess == secretNum):

break

elif(abs(secretNum-userGuess)<=2):

print("You're close!")

else:

print("You're pretty far off")

userGuess = int(input("Guess again >> "))

print("Correct! I was thinking of " + str(secretNum))

### **time.sleep() Arguments**

**secs** - The number of seconds the Python program should pause execution. This argument should be either an int or a float.

import time

# Wait for 5 seconds

time.sleep(5)

# Wait for 300 milliseconds

# .3 can also be used

time.sleep(.300)

**Comparing strings**

Strings are case-sensitive, so you need to be somewhat careful when comparing them.

Use **upper()** or **lower()** before comparing to strings.

This is usefule when asking the user to enter a data that you need to compare to.

strName = "bob"

if (strName == "Bob"): print("true")

else: print("false")

# Notice this code will result in “false”, because the of the Capital “B”.

# It is common to covert data to a upper or lower case for comparisons

if (strName.lower() == "Bob".lower()): print("true")

else: print("false")

# This code will print “true”, even though there is a Capital “B” in one of the strings.

**Class**

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods.A **Class** is like an object constructor, or a "blueprint" for creating objects. It’s like creating your own module to use later. For example, when you import modules “time” and “random”.

**Example**

class MyClass:  
x = 5

point = MyClass()  
print(p1.x)

**Output:**

5

**open()**

**Writing Data to a File**

Writing data to files is a practical use of while loops and if statements. Here is an example of how to create and write to a text file:

**open(file, mode)**

**Example – At the start, to create a new text file even if one does not exit yet**

open(“textfile.txt”, "a")

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *file* | The path and name of the file |
| *mode* | A string, define which mode you want to open the file in:  "r" - Read - Default value. Opens a file for reading, error if the file does not exist  "a" - Append - Opens a file for appending, creates the file if it does not exist  "w" - Write - Opens a file for writing, creates the file if it does not exist  "x" - Create - Creates the specified file, returns an error if the file exist  In addition you can specify if the file should be handled as binary or text mode  "t" - Text - Default value. Text mode  "b" - Binary - Binary mode (e.g. images)  “r+'” - Open for reading and writing. The stream is positioned at the  beginning of the file.  “w+” - Open for reading and writing. The file is created if it does not  exist, otherwise it is truncated. The stream is positioned at  the beginning of the file.  “a+” - Open for reading and writing. The file is created if it does not  exist. The stream is positioned at the end of the file. Subse-  quent writes to the file will always end up at the then current  end of file, irrespective of any intervening [**fseek(3)**](http://www.manpagez.com/man/3/fseek/) or similar. |

Links:

<http://www.manpagez.com/man/3/fopen/>

**Example 1**

~~objFile =~~ **~~open~~**~~("C:\\\_PythonClass\\TestData.txt", "a")~~

# this code should be written like this instead so that the path is not a hard dpath

objFile = **open**("TestData.txt", "a") # creates a new txt file called TestData; objFile is a newly created variable

objFile.**write**(input("Enter your data: ") + "\n") # writes data in the file

objFile.**close**()

**Example 2**

strFName = "Bob"

strLName = "Smith"

happy = open("MyData.txt", "w")

happy.write(strFName + '\t' + strLName + '\n')

happy.write(strFName + ',' + strLName + '\n')

happy.close()

**Output**

A newly create text file in the same folder as the python script will have this data in it:

Bob Smith

Bob,Smith

**None**

You can assign a variable to **None**

**Example**

Var = None

if(var):

print(T”)

else:

print(“F”) # Output will be “F”.

**import sys**

**You have to run programs like this from a command shell because you have to include the arguments in**

**Example**

import sys

if(len(sys.argv) > 1):

intArg1 = int( sys.argv[1] ) # Get Argument 1

intArg2 = int( sys.argv[2] ) # Get Argument 2

strData = str(intArg1 + intArg2) # Perform some Processing

#print("The Sum of the first and second arguments is: " + strData)

print(intArg1)

print(intArg2)

print(strData)

else:

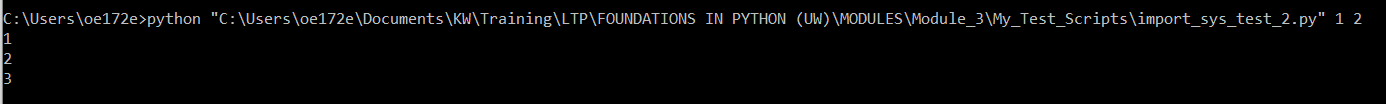
print("This script requires two argument to run")

print("Ex: MyScript Arg1 Arg2")

For output, go to Command Shell and type in:

python Path-of-file aurgument\_1 argument\_2

Example:



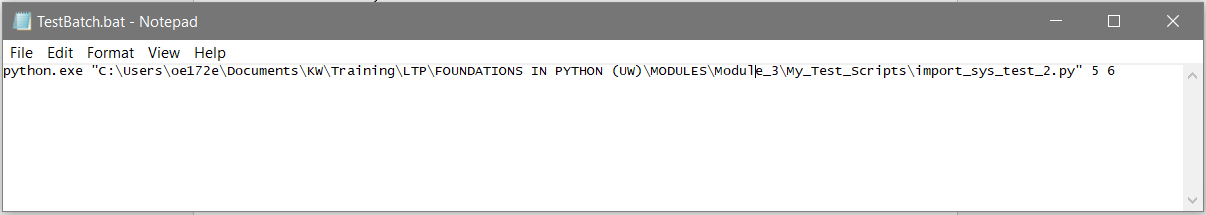
**Output:**

1

2

3

# This is a great feature to run automation usng batch files. Below is how simple the btach file would look like:



**while not**

# it loops when nothing is entered; loop and continue to ask the user for username and password

username = **""  
while not** username:  
 username = input(**"Username: "**)

password = **""  
while not** password:  
 password = input(**"Password: "**)

**if** username == **"M.Dawson" and** password == **"secret"**:  
 print(**"Hi, Mike."**)  
 security = 5

**elif** username == **"S.Meier" and** password == **"civilization"**:  
 print(**"Hey, Sid."**)  
 security = 3

**elif** username == **"guest" or** password == **"guest"**:  
 print(**"Welcome, guest."**)  
 security = 1

**else**:  
 print(**"Login failed. You're not so exclusive.\n"**)  
input(**"\n\nPress the enter key to exit."**)

**int(input())**

# when you ask for an number input from the user and want to make sure the input is an interger, just use “int()” and place the input inside the brackets.

[**Module 4**](#_top)

**Adding to a Tuple**

# To **add data to a tuple,** you use the rather **odd syntax** shown here:

**Example:**

tplData = ("1","2","3")

print(tplData)

tplMoreData = ("4"), #Note that the comma at the end makes this a tuple!!!

tplData += tplMoreData # One tuple added to another works fine!

print(tplData) #Actually another new tuple (just like strings!)

# tplData = tplData + '4' # Adding a string to a tuple does NOT work!

# tplData[2] = 4 # Modifying value in a tuple does NOT work!

**Output:**

**('1', '2', '3')**

**('1', '2', '3', '4')**

**In Operator**

# The "*in"* operator searches though a sequence and return a Boolean if found.

**for in end=**

**for variable in string**

**for loop**

# like a while loop but newer. More efficient; no need to use conditions. Use as an iterator.

**Example 1**

word = input("Enter a word: ") # word is declaed a string variable

print("\nHere's each letter in your word:")

**for** letter **in** word: # here, letter is declared as a character variable

print(letter)

**Output**

Enter a word: blah

Here's each letter in your word:

b

l

a

h

**Example 2**

strData = "ABC"

for i in strData: # i is declared as a character variable here

print(i)

**Output**

A

B

C

**Example 3** # using **for** with **end=**

strData = “ABC”

for i in strData:

print(i, end=’!’)

**Output**

A!B!C!

**Example 4**

bob = (1, 'Bob Smith', 'BSmith@Hotmail.com')

sue = (2, 'Sue Jones', 'SueJ@Yahoo.com')

joe = (3, 'Joe James', 'JoeJames@Gmail.com')

names = bob, sue, joe

for i in names: # notice that we use “names”. But look at Example 5.

print(i)

**Output:**

(1, 'Bob Smith', 'BSmith@Hotmail.com')

(2, 'Sue Jones', 'SueJ@Yahoo.com')

(3, 'Joe James', 'JoeJames@Gmail.com')

**Example 5**

for i in bob: # this time we use “bob” instead of “names”. See output difference.

print(i)

**Output:**

1

Bob Smith

BSmith@Hotmail.com

**range()**

**range(a, b, c)**

**# a = starting number**

**# b = ending number**

**# c = counts by c**

**Example 1**

# counts from 0 to 9

for i in range(10): # counts from 0 to 9

print(i, end=" ") # the space “ “ is what you want to add between each number

**Example 2**

# counting by fives

for i in range(0, 50, 5): # starts at 0, ends before 50, counts by 5

print(i, end=" ")

**Example 3**

# counting backwards

for i in range(10, 0, -1): # starts at 10, ends before 0, counts backwards by 1

print(i, end=" ")

**.strip()**

**Example 1**

strTask = str(input(**"Enter a task: "**))**.strip()**

**Example 2**

**for** line **in** objFile:  
    strData = line.split(**","**)  
    dicRow = {**"Task"**: strData[0]**.strip()**, **"Priority"**: strData[1]**.strip()**}  
    lstTable.append(dicRow)  
objFile.close()

**String Methods**

**Example**

strData = " tEsT Data "

print(strData.lower())

print(strData.upper())

print(strData.replace(" ", "--"))

print(strData.strip()) # returns a copy of the string with both leading and trailing characters removed. It’s great for removing brackets from a list.

**Example**

text = " hello World "

print(text) # prints hello World

print(text.strip()) # prints hello World

print(" Hello World ") # prints hello World

print(" Hello World ".strip()) # prints hello World

print(strData.isalpha())

# Returns :

# 1.True- If all characters in the string are alphabet.

# 2.False- If the string contains 1 or more non-alphabets.

strData = '1,2,3'

lstData = strData.split('2')

print(lstData[0], lstData[1], lstData[2], sep='|')

**Output**

test data

TEST DATA

------tEsT------Data----

tEsT Data

False

123

split()

# the split() method splits a string into a list.

# You can specify the separator, default separator is any whitespace.

# string.split(separator, max)

# Parameter Description

# separator Optional. Specifies the separator to use when splitting the string. Default value is a whitespace

# max Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences"

**Example 1**

txt = "apple#banana#cherry#orange"

x = txt.split("#")

print(x) # Prints: ['apple', 'banana', 'cherry', 'orange']

**Example 2**

txt = "hello, my name is Peter, I am 26 years old"

x = txt.split(", ")

print(x) # Prints: ['hello', 'my name is Peter', 'I am 26 years old']

# note the white space after the comma is included in the split

**Example 3**

txt = "welcome to the jungle"

x = txt.split()

print(x) # Prints: ['welcome', 'to', 'the', 'jungle']

**Example 4**

txt = "apple#banana#cherry#orange"

# setting the max parameter to 1, will return a list with 2 elements!

x = txt.split("#", 1)

print(x) # Prints: ['apple', 'banana#cherry#orange']

**Example 5**

txt = "apple#banana#cherry#orange"

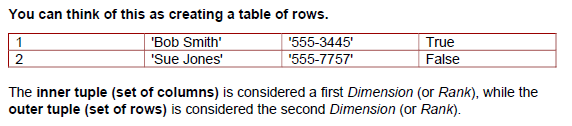
x = txt.split("#", 2)

print(x) # Prints: ['apple', 'banana', 'cherry#orange']

**col row**

# print a row as a column

# display columns and rows in a tuple



**Example 1**

bob = (1,"Bob Smith","555-3445", True)

sue = 2,"Sue Jones","555-7757", False

table = (bob, sue)

for row in table:

for col in row:

print(col)

**Output:**

1

Bob Smith

555-3445

True

2

Sue Jones

555-7757

False

**[MODULE 5](#_top)**

**Dictionary**

**Writing dictionary to text file**

**Example 1**

dict = {‘id’ : '1', 'name' : 'Bob', 'Email' : 'bob@gmail.com'}

f = open("textfile.txt","w") # “w” is write

f.write(str(dict))

f.close()

**OUTPUT**

Data in of text file,”textfile.txt”

{'id': '1', 'name': 'Bob', 'Email': 'bob@gmail.com'}

**Example 2 (compare to Example 1)**

dict = {"id" : "1", "name" : "Bob", "email" : "bob@gmail.com"}

f = open("textfile3.txt", "w")

f.write(dict["id"] + ',' + dict["name"] + ',' + dict["email"] + ‘\n’)

f.close()

**OUTPUT**

Data in of text file,”textfile3.txt”

1,Bob,bob@gmail.com

**Example 3 – Add a table to a text file**

objFile = open(textfile, "w")

for objRow in lstTable:

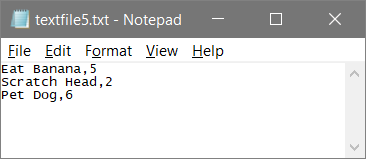
objFile.write(objRow["Task"] + "," + objRow["Priority"] + "\n")

objFile.close()

print("Data Saved!")

**Loading / Pulling / Reading from a text file and displaying the data in the console as a dictionary**

**Example 1 – displaying items from textfile to a table (Module 5, Listing 8)**



# Existing text file in the folder

# Declare my variables

objFile = **None**

dicRow = {}

lstRow = []

lstTable = []

open(objFile, "a")

objFile = open(objFile, "r")

for row in objFile:

strData = row.split(",") # Returns a list!

dicRow = {"task": strData[0], "priority": strData[1]}

lstTable.append(dicRow)

objFile.close()

objFile = open('textfile5.txt', "r")

for row in objFile:

lstRow = row.split(",")

# takes a string with and seperates the items with "," between and returns as a list

dicRow = {"task": lstRow[0], "priority":lstRow[1].strip()}

# takes the list and puts it in a dictionary. the strip() will remove the '\n'

lstTable.append(dicRow)

# takes each dictionary and puts it in a table. The table is not displayed in the console but used to save to a textfile.

objFile.close()

**Output**

No output for the script above because it only loads the text file data into a table of dictionary list. But to display the list of items in the table, use this script below.

# note that you don’t need to open the text file and pull the data from the text file again because lstTable already contains the data, and you just need to display it.

for dicRow in lstTable:

print(dicRow)

**Output 1**

{'Task': 'Eat Banana', 'Priority': '6'}

{'Task': 'Scratch Head', 'Priority': '3'}

{'Task': 'Pet Dog', 'Priority': '6'}

**Output 2**

To display a different way without the curly brackets, use the script below instead:

for row in lstTable:

print(row['task'] + ", " + row['priority'])

**Output**

Eat Banana,5

Scratch Head,2

Pet Dog,6

**Example - To display a table of dictionaries**

a = {**"keyA"**: 123, **"keyB"**: 456, **"keyC"**: 789}  
b = {**"keyA"**: 321, **"keyB"**: 654, **"keyC"**: 987}  
table = [a,b]

**for** objRow **in** table:  
 print(objRow)

**Output**

{'keyA': 123, 'keyB': 456, 'keyC': 789}  
{'keyA': 321, 'keyB': 654, 'keyC': 987}

**Dictionary has helpful functions like items()**

**Example 1**

for myKey, myValue in dicRow.items

print(myKey, " = ", myValue)

**Output**

item = End Table

value = $1590

**Example 2**

# unpacking the elements with the items() function

*# Create a data structure*dicRow1 = {**"ID"**:1,**"Name"**:**"Bob Smith"**, **"Email"**:**"**[**BSmith@Hotmail.com**](mailto:BSmith@Hotmail.com)**"**}  
dicRow2 = {**"ID"**:**"2"**,**"Name"**:**"Sue Jones"**, **"Email"**:**"**[**SueJ@Yahoo.com**](mailto:SueJ@Yahoo.com)**"**}  
lstTable = [dicRow1, dicRow2]

*# Process the data*

*# prints with square brackets*print(**"\n--- items in the list 'Table'"**)  
print(lstTable)

*# prints with square bracket***for** objRow **in** lstTable:  
 print(objRow)

*# unpacking the elements with the items() function*print(**"\n--- Unpacking the elements with the items() function"**)  
**for** myKey, myValue **in** dicRow1.items():  
 print(myKey, **" = "**, myValue )

print("\n--- Displaying only the values()")

print(dicRow1.values())

print("\n--- Displaying only the keys()")

print(dicRow1.keys())

**Appending a table with user inputs**

# Get User Input

**Example**

strID = input("Enter an ID: ")

strName = input("Enter an Name: ")

strEmail = input("Enter an Email: ")

dicRow = {"id":strID,"name":strName, "email":strEmail}

lstTable.append(dicRow)

**pop()**

**To remove a row of items from a table list**

**Example, to remove “Scratch Head” from table below:**

Eat Banana,5

Scratch Head,2

Pet Dog,6

**Example**

strTask = input("Enter an task you would like to remove: ")

for var in range(len(lstTable) -1, -1, -1):

if lstTable[var]["Task"] == strTask:

lstTable.pop(var)

**Output**

User will enter “Scratch Head” when prompted and the new table will now be:

Eat Banana,5

Pet Dog,6

**Printing a table dictionary**

**for** objRow **in lstTable**:

print(objRow)

**if not**

**not() # not() is not a function but I’m just using it as a bookmark**

# if lstTable has no data, it wiil displays “No data in list”.

Look at Step 3 in file “Assigment05\_Starter.py” at:

C:\Users\oe172e\Documents\KW\Training\LTP\FOUNDATIONS IN PYTHON (UW)\MODULES\Module\_5\GitHub Peer Review\Tony Le

Also look at:

https://www.jquery-az.com/4-demos-python-if-not-and-not-in-operator/

**Example**

if not lstTable:

print("No data in list")

else:

print(lstTable)

continue

**,end =" "** # keywords: sentence, line, ending, escape

# ends the output with a <space>  and continues additional print stantements on the same line

**Example**

print(**"Welcome to"** , end = **' '**)  
print(**"GeeksforGeeks"**, end = **' & '**)  
print(**"Happy"**)

# note the space between the ' and '.

**Output**

Welcome to GeeksforGeeks & Happy

[**MODULE 6**](#_top)

**Functions**

Functions are a way of **grouping one or more statements**. In Python, you must **define** a function **before you** can use code to **call the function**.

**Listing 1.**

**def**

**Example**

def myfunction():

print("Hello World")

myfunction()

**Output**

Hello World

**Parameters (or arguments)**

Optionally, functions can have **parameters**. These **allow you to pass values into the function for processing**. **Values passed into parameters are officially called “arguments,”** but it's common for people to call them parameters too.

**Listing 2.**

**Example 0.5**

**# the argument “message” inside the tuple is a LOCAL variable within function “display”.**

**# you can use this local variable as an user input later.**

def display(message):

print(message)

display("Here's a message for you.\n")

**Output**

Here is a message for you.

**Example 1 – using 1 parameter**

# Define the function

def myfunction(myparameter):

print("The parameter is: " + myparameter)

# Call the function

myfunction("blah")

myfunction("2")

**Output**

The parameter is: blah

The parameter is: 2

**Example 2 – using 2 parameters**

# Define the function

def myfunction(par1, par2):

print("The parameters are: " + par1 + " and " + par2)

# Call the function

myfunction("hee","haa")

**Output**

The parameters are: hee and haa

**Example 2.5**

# using 2 paarmeters in a function

def birthday(a = “blah”, b = “0”):

# blah and 0 are default parameters to avoid error messages if user doesn’t provide and input

print("Happy birthday,", a, "!", " I hear you're", b, "today.\n")

name = input("Name: ")

age = input("Age: ")

birthday(name, age)

**Output**

Name: Kevin

Age: 44

Happy birthday, Kevin ! I hear you're 44 today.

**Example 3 – Listing 3: 2 parameters that calculates a sum**

# Define the function

def AddValues(value1, value2):

fltAnswer = value1 + value2

print("The Sum of the values is: " + str(fltAnswer))

# Call the function

AddValues(10, 5)

**Output**

The Sum of the values is: 15

**del**

**Example**

a = **"Hello "**

b = **"World"**

**del** b

print(a + b)

**Output**

NameError: name 'b' is not defined

**CLASS**

**class** Math():

*# creates a class with two functions; one function to add, and another to subtract*

@staticmethod  
 **def** Add(value1=0.0, value2=0.0):  
 **return** float(value1 + value2)

@staticmethod  
 **def** Subtract(value1=0, value2=0):  
 **return** float(value1 - value2)

**print**(Math.Add(1,2))  
**print**(Math.Subtract(10,8))

**Output**

3.0

2.0

**global variable**

In a def function, if a variable is intended to be a Global Variable, then you need to assign it as such by:

global var

if not, Python will consider it a a local variable within the def function only and you cannot use this variable outside of the function.

**concatenate strings and floats**

num = 2  
**print**(**"The number is %s"** % num) *# prints value as a string***print**(**"The number is %f"** % num) *# prints value as a float***print**(**"The number is %.3f"** % num) *# print value as float with 3 decimal places*

**[MODULE 7](#_top)**

**Pickling**

[**https://www.datacamp.com/community/tutorials/pickle-python-tutorial**](https://www.datacamp.com/community/tutorials/pickle-python-tutorial)

### **What is pickling?**

**In a nutshell: Saves data in a binary file .DAT**

Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object. Pickling is not to be confused with compression! The former is the conversion of an object from one representation (data in Random Access Memory (RAM)) to another (text on disk), while the latter is the process of encoding data with fewer bits, in order to save disk space.

### **What Can You Do With pickle?**

Pickling is useful for applications where you need some degree of persistency in your data. Your program's state data can be saved to disk, so you can continue working on it later on. It can also be used to send data over a Transmission Control Protocol (TCP) or socket connection, or to store python objects in a database. Pickle is very useful for when you're working with machine learning algorithms, where you want to save them to be able to make new predictions at a later time, without having to rewrite everything or train the model all over again.

### **When Not To Use pickle**

If you want to use data across different programming languages, pickle is not recommended. Its protocol is specific to Python, thus, cross-language compatibility is not guaranteed. The same holds for different versions of Python itself. Unpickling a file that was pickled in a different version of Python may not always work properly, so you have to make sure that you're using the same version and perform an update if necessary. You should also try not to unpickle data from an untrusted source. Malicious code inside the file might be executed upon unpickling.

**Example – To pickle a dictionary**

import **pickle**

dogs\_dict = { 'Ozzy': 3, 'Filou': 8, 'Luna': 5, 'Skippy': 10, 'Barco': 12, 'Balou': 9, 'Laika': 16 }

filename = 'dogs'

outfile = open(filename,'wb') # w means writing to file, and b refers to binary mode.

**pickle.dump**(dogs\_dict,outfile)

outfile.close()

**Output**

A new .DAT binary file named dogs should have appeared in the same directory as your Python script (unless you specified a file path as file name).

**Example – To unplickle the dictionary**

import pickle

filename = 'dogs'

infile = open(filename,'rb')

new\_dict = pickle.load(infile)

infile.close()

print(new\_dict)

print(type(new\_dict))

**Output**

{'Ozzy': 3, 'Filou': 8, 'Luna': 5, 'Skippy': 10, 'Barco': 12, 'Balou': 9, 'Laika': 16}

<class 'dict'>

**Exception Handling (Structured Error Handling)**

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Great links:

<https://www.python-course.eu/python3_exception_handling.php>

<https://www.pythonforbeginners.com/error-handling/exception-handling-in-python>

<https://www.w3schools.com/python/python_try_except.asp>

**Example 1**

try:

print(x)

except:

print("An exception occured")

**Output**

An exception occured

**Example 2**

try:

print("Hello")

print(x)

except:

print("Something went wrong")

else:

print("Nothing went wrong")

**Output**

Hello

Something went wrong

**Example 3**

try:

print(1/0)

except ZeroDivisionError:

print("You can't divide by zero, you're silly.")

**Output**

You can't divide by zero, you're silly.

**Example 4**

while True:

try:

n = int(input("Please enter an integer: "))

break

except ValueError:

print("No valid integer! Please try again ...")

print("Great, you successfully entered an integer!")

**Output**

Please enter an integer: a

No valid integer! Please try again ...

Please enter an integer: 4.0

No valid integer! Please try again ...

Please enter an integer: 1

Great, you successfully entered an integer!

**Using the Exception Class**

"Exception" is a built-in python **class used to hold information about an error**. Python **automatically creates an Exception object when an error occurs**. The Exception object automatically fills **with information about the error** that caused the exception.

You can **capture the Exception object in the except section** of a try-except block and extract the error messages (Listing 12).

**Example**

try:

quotient = 5/0

print(quotient)

except Exception as e:

print(e)

print(type(e))

print(e.\_\_doc\_\_)

print(e.\_\_str\_\_())

**Output**

division by zero

<class 'ZeroDivisionError'>

Second argument to a division or modulo operation was zero.

division by zero

**List of some common exceptions errors**

IOError

If the file cannot be opened.

ImportError

If python cannot find the module

ValueError

Raised when a built-in operation or function receives an argument that has the

right type but an inappropriate value

KeyboardInterrupt

Raised when the user hits the interrupt key (normally Control-C or Delete)

EOFError

Raised when one of the built-in functions (input() or raw\_input()) hits an

end-of-file condition (EOF) without reading any data

# List of Built-in Exceptions

<https://docs.python.org/3/library/exceptions.html#bltin-exceptions>

**[MODULE 8](#_top)**

[**https://youtu.be/ZnTabY0Z-XE?t=1701**](https://youtu.be/ZnTabY0Z-XE?t=1701)

**Good Websites on Classes:**

<https://realpython.com/python3-object-oriented-programming/>

<https://www.w3schools.com/python/python_classes.asp>

**Naming convention In a Regular Script Naming Convention in a Class Real Life Name**

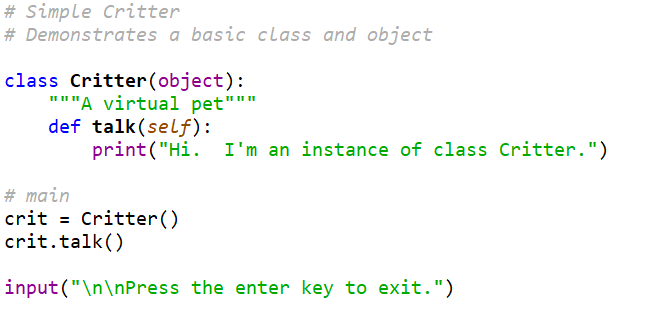
Data (Variable or Constant) -------------------------- Fields (aka Attributes / Properties) ------ Characteristics

Functions ------------------------------------------------- Methods (aka Procedures) ------------------Behaviors

Instantiated ------------------------------------ Created

Instance --------------------------- --------- Object

Class ----------------------------------------- Blueprint



**talk()** is a method

**self** is an parameter

Note: **Fields**, **attributes** and **properties** are 3 different ways to manage data.

A Class is an objects, but it’s a design for one.

**You can use your Class directly or indirectly.**

The problem with using a Class directly is that you can only have one Person in this case.

The advantage of using Class indirectly by using multiple object instants.

Using a class **directly**:

Class Person.strFirstName = “Bob”

print(Person.strFirstName)

Using a class **indirectly**:

objP1 = Person.strFirstName # objP1 is an object instants

print(objP1.strFirstName)

A Standard Class Pattern

**class** MyClassName:

# -- **Fields** –

# -- **Constructor** –

# -- **Attributes** –

# -- **Properties** –

# -- **Methods** –

**Constructors**

**Definitions**

* **Classes** typically have Fields, Constructors, Properties, and Methods.
* **Fields** are the data members of a class. **Fields** are created using variables and constants.
* **Constructors** are special **methods** (functions) that automatically runs when you create an object from the class. **Constructors** are often used to set the *initial values* of **Field** data. Constructors only run *once*; when a new object instance of a class is created!

first\_name is a parameter

“Sue” is an argument

**Attributes** are "virtual"fields that hold internal data

**Note:** One problem with **Fields or Attributes** is that they **are just variables**. You do not have much control over what data goes into them unless you **write specific code to validate values before they are assigned**. **To help** with that, you can **use special methods (functions) called Properties**.

Best Practices: “self” in the code below can be changed but not recommended

Example 1

# Module 08 Listing 02 *– A class with a constructor*

class Person:

# --Fields--

strFirstName = ""

# -- Constructor --

def \_\_init\_\_(self, first\_name =''):

#-- Attributes --

self.strFirstName = first\_name

# -- Properties --

# -- Methods --

# --End of class--

# --- Use the class ----

objP1 = Person() # with only no argument

objP2 = Person(first\_name="Sue") # with the parameter and argument

print(objP1.strFirstName) # will be empty

print("-------------")

print(objP2.strFirstName) # will have first name

**Output**

-------------

Sue

Example 2

**The same code as above but with all the comments removed to help with understanding**

class Person:

strFirstName = ""

def \_\_init\_\_(self, first\_name =''):

self.strFirstName = first\_name

objP1 = Person()

objP2 = Person(first\_name="Sue")

print(objP1.strFirstName)

print("-------------")

print(objP2.strFirstName)

**Output**

-------------

Sue

Example 3

class Person:

strFirstName = "Default"

def \_\_init\_\_(self, first\_name =''):

self.strFirstName = first\_name

objP1 = Person(first\_name="Bob")

objP2 = Person(first\_name="Sue")

print(objP1.strFirstName)

print("-------------")

print(objP2.strFirstName)

print("-------------")

print(Person.strFirstName)

**Output**

Bob

-------------

Sue

-------------

Default

**Example 4**

From link: <https://www.youtube.com/watch?v=ZDa-Z5JzLYM&list=PL-osiE80TeTsqhIuOqKhwlXsIBIdSeYtc&index=2&t=0s>

class Employee:

def \_\_init\_\_ (*self*, first, last):

*self*.first = first

*self*.last = last

*self*.price = price

def fullname(self):

return '{} {} {:.2f}'.format(*self*.first, *self*.last)

emp1 = Employee("Apple", "Red", 50)

emp2 = Employee("Banana", "Yellow", 75)

# slower and more cumbersome way

print(emp1.first, emp1.last, emp1.price)

print(emp2.first, emp2.last, emp2.price)

# faster way by creating a special method called “fullname”

print(emp1.fullname()) # one way of using fullname method

print(Employee.fullname(emp1)) # another way of using fullname method

**Output**

Apple Red 50.00

Banana Yellow 75.00

Apple Red 50.00

Apple Red 75.00

***Note:*** *Since* ***constructors are a specialized function****, so you use them as a function by passing arguments into the parameters. However,* ***remember****, they* ***only run once; when a new object instance of a class is created****!*

Destructors

Another special method is the "Destructor." These **automatically run when an object instance goes is removed from memory**. They are used to "clean up" any resources that are not needed once the object is gone. In Python, most of the resources are "self-cleaning," and so **you do not often see these in classes like you do Constructors**.

Destructors are considered an **advanced feature** and should be used with care. We **do not go into them in this course**, but their code looks like this.

def **\_\_del\_\_**(self):

""" automatically called when object is destroyed"""

*# TODO: Add some "cleanup" code*

**special methods: \_\_str\_\_ and \_\_repr\_\_**

<https://pythonprogramming.net/__str__-__repr__-intermediate-python-tutorial/>

<https://dbader.org/blog/python-repr-vs-str>    
The \_\_str\_\_ method is useful for a string representation of the object.  
The repr method is really meant to be just for developers, and more for debugging than actual use of the module.

**Example 1**

class Car:  
    def \_\_init\_\_(self, color, mileage):  
        self.color = color  
        self.mileage = mileage

car1 = Car("red"

print

    def \_\_str\_\_(self):  
        return f'blah {self.color} {self.mileage}'  
      
    def \_\_repr\_\_(self):  
        return f'meh {self.color} {self.mileage}'  
      
car1 = Car("red", 50)  
  
print(car1)  
print(str(car1))  
print(repr(car1))

**Output**blah red 50  
blah red 50  
meh red 50

**Best Practice:**

**Add this code below for \_\_repr\_\_ at the very least to all Classes, ideally also include \_\_str\_\_:**

class Car:  
    def \_\_init\_\_(self, color="gray", mileage="0"):  
        self.color = color  
        self.mileage = mileage

    def \_\_repr\_\_(self):

        return '{self.\_\_class\_\_.\_\_name\_\_}({self.color}, {self.mileage})'.format(self=self)

**[MODULE 9](#_top)**

SAF

def standalone\_function():

print("Called standalone\_function")

class MyData:

def \_\_init\_\_(self):

print("Created MyData object")

class FileProcessor:

@staticmethod

def process\_data():

print("Called process\_data")

class IO:

@staticmethod

def print\_data():

print("Called print\_data")

**\_\_name\_\_**

Weblinks: <https://www.afternerd.com/blog/python-__name__-__main__/>

Use this code in all modules that are not your applications main module:

if \_\_name\_\_ == "\_\_main\_\_":

raise Exception("This file was not created to be imported")

**[GLOSSARY](#_top)**

[**Attributes**](https://en.wikipedia.org/wiki/Attribute_(computing))

A specification that defines a [property](https://en.wikipedia.org/wiki/Property_(programming)) of an [object](https://en.wikipedia.org/wiki/Object_(computer_science)), element, or file. It may also refer to or set the specific [value](https://en.wikipedia.org/wiki/Value_(computer_science)) for a given instance of such.

For clarity, attributes should more correctly be considered [metadata](https://en.wikipedia.org/wiki/Metadata)

**Functions**

Statements are often grouped into **Functions** (also known as **methods** or **sub-procedures**) Example of a function is:

def DemoMethod()

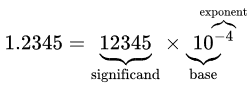
code, code, code

#end DemoMethod

**Floating Point Number**

Represented like this:





**Interger**

An integer (more commonly called an int) is a number without a decimal point. A float is a floating-point number, which means it is a number that has a decimal place. Floats are used when more precision is needed.

**Binary Digit (BIT)**

One binary digit (0 or 1) is referred to as a *bit. O*ne bit can be implemented by one switch.

**Byte**

A byte is implemented with eight switches

## **Class**

AKA Module

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods.A **Class** is like an object constructor, or a "blueprint" for creating objects. It’s like creating your own module to use later. For example, when you import modules “time” and “random”.

Creating a new class creates a new type of object, allowing new instances of that type to be made.

**Example**

class ClassName:

<statement-1>

.

.

.

<statement-N>

<https://docs.python.org/3/tutorial/classes.html>

# *Py*Format

[**https://pyformat.info/**](https://pyformat.info/)

[**REFERENCES**](#_top)

# FROM MODULE 1

**Readings in Module 1**

1. Chapter 1 ofPython Programming for the Absolute Beginner(Third Edition) by Michael Dawson.
2. PROGRAMMING WITH PYTHON *By Randal Root*

(In the downloaded zip folder from the Course Canvas)

1. Style Guide for Python Code

# <https://www.python.org/dev/peps/pep-0008/>

1. Command Line

<https://www.computerhope.com/jargon/c/commandi.htm>

1. Python Getting Started

<https://www.w3schools.com/python/python_getstarted.asp>

**Videos Watched in Module 1**

1. **Introduction to Python module 1 of 10 by Professor Randal Root**

<https://www.youtube.com/watch?v=pa9GRFAYm4s&feature=youtu.be>

1. **Module Videos by Professor Randal Root**

<https://www.youtube.com/playlist?list=PLfycUyp06LG9OVfidlxwfjz7JYRhq83PN>

# Style Guide for Python Code

# <https://www.python.org/dev/peps/pep-0008/>

1. **Command Line**

<https://www.computerhope.com/jargon/c/commandi.htm>

# Python Getting Started

<https://www.w3schools.com/python/python_getstarted.asp>

1. **Module Videos by Professor Randal Root**

<https://www.youtube.com/playlist?list=PLfycUyp06LG9OVfidlxwfjz7JYRhq83PN>

1. **Video - Using Python interactively**

<https://realpython.com/lessons/running-python-code-interactively/>

1. **Video - Print and Input**

<https://youtu.be/FhoASwgvZHk>

1. **Video - Python Tutorial for Beginners 1: Install and Setup for Mac and Windows** <https://www.youtube.com/watch?v=YYXdXT2l-Gg&vl=en>
2. **Video - Introduction to Batch files (Windows)**

<https://youtu.be/rhV4L3T3BMc>

1. **Video - Introduction to Bash files (Mac)**

<https://youtu.be/nZqi3BqqeqI>

1. **Video - Creating Professional Documents**

<https://www.youtube.com/watch?v=9ojhSW9ljjo&feature=youtu.be>

1. **This link was in the Module 01 youtube video but did not work. It’s on how to write professional papers**

http://jerz.sentonhill.edu/writing/academic1/mla-style-papers/

[**https://youtu.be/wAIylkABVDc?t=2625**](https://youtu.be/wAIylkABVDc?t=2625)