

1. Introduction

Goal of Project:

Learn as much as possible about basic python

TABLE OF CONTENTS

- [1. Introduction](#)
- [2. Body](#)
 - [2.1. Basic Information](#)
 - [2.1.1. Building and Compiling](#)
 - [2.1.1.1. Command Line Python Interpreter](#)
 - [2.1.1.2. .py Files](#)
 - [2.1.1.3. Advanced Interpreter](#)
 - [2.2. Syntax](#)
 - [2.2.1. White Space](#)
 - [2.3. Variables](#)
 - [2.3.1. Built in Scalar Types](#)
 - [2.3.2. Data Structures](#)
 - [2.3.2.1. Examples](#)
 - [2.4. Math](#)
 - [2.5. Logic and Loops](#)
 - [2.5.1. Conditional Statments](#)
 - [2.5.2. For loops](#)
 - [2.5.3. While Loops](#)
 - [2.6. Input and Output](#)
 - [2.6.1. Screen](#)
 - [2.6.2. Files](#)
 - [2.7. Functions](#)
 - [2.8. Imports](#)
- [3. Code Project](#)
 - [3.1. Sameple output](#)
 - [3.2. Code](#)

2. Body

Need to show	How to show
Compiling and Debugging	Console
	.Py File
	Jupyter Labs
Basic programing Structures	Variables

Need to show	How to show
	Operations
	Whitespace
	imports
	Scope
Data structures	Arrays
Functions	
Input	
Output	

2.1. Basic Information

This solution simply presents the basic information of python AND will not have any asociated code.

2.1.1. Building and Compiling

Runing python is different to other languages because you can run python through multiple means.

2.1.1.1. Command Line Python Interpreter

Python can be run in the command line. To get started type **py** into the command line.

You will then see the following text in your command line, informing you have just launched the command line python interpreter.

```
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
```

Warning:

This Python interpreter is in a conda environment, but the environment has not been activated. Libraries may fail to load. To activate this environment please see <https://conda.io/activation>

Type "help", "copyright", "credits" or "license" for more information.

If you did not see this text it is advised that you check your instalation of python.

Here you can type any line of code such as

```
>>>print("Hello World")  
Hello World
```

2.1.1.2. .py Files

Another way to work with python is to write the code into a text file with the extension .py. Any .py file can be quickly read.

One example is the code below

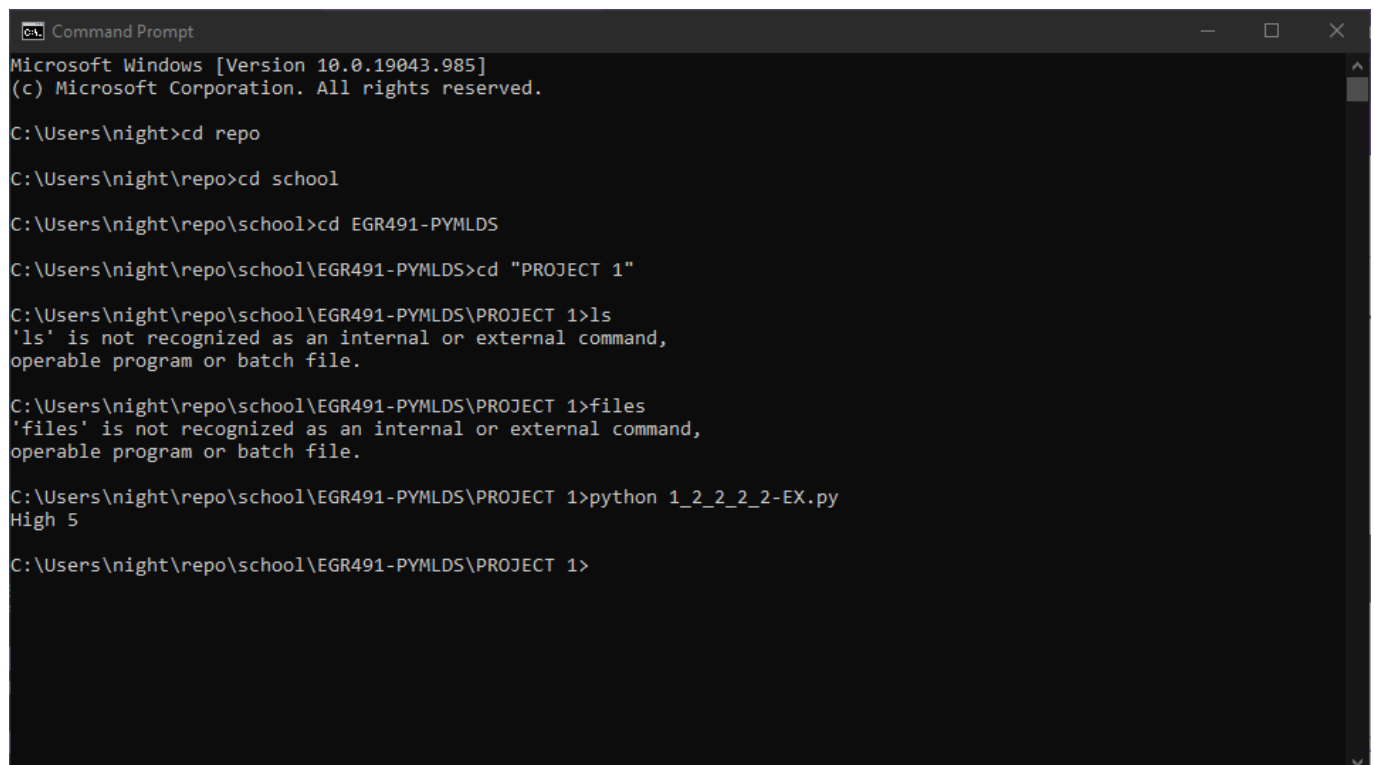
```
x = 0
x += 1
x *= 5
print("High", 5)
```

This will output

```
High 5
```

There are two ways to run these files,

1. Open the File in an IDE such as Jupyter Lab IDE or Visual Studio Code
 1. Right Click the file and select OPEN WITH
 2. Select the IDE of your choice
 3. These IDE's will allow you to run the file. In VSCode you will need to install the proper extension to run python files.
2. Run the files via command line.
 1. Locate the file in command line (you can use `cd < folder name >` if it is not in your root directory)
 2. Type `python < file name >`
 3. A python interpreter will launch showing your output



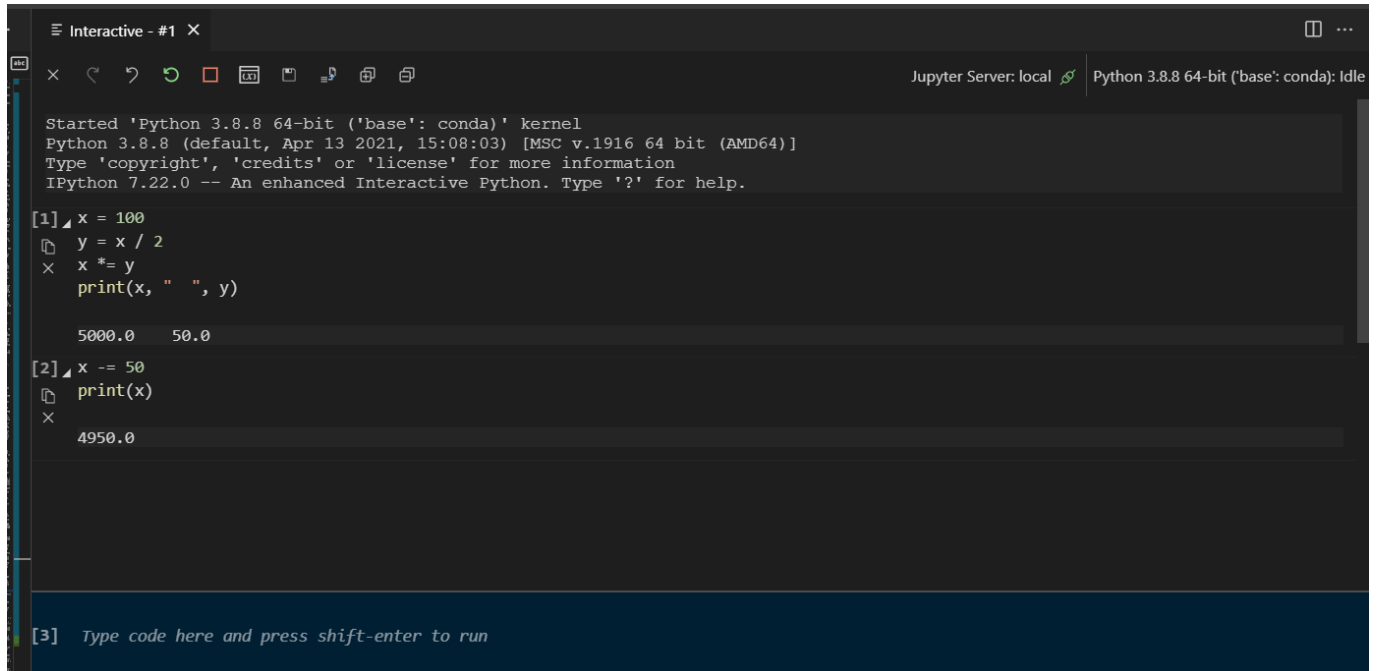
```
Command Prompt
Microsoft Windows [Version 10.0.19043.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\night>cd repo
C:\Users\night\repo>cd school
C:\Users\night\repo\school>cd EGR491-PYMLDS
C:\Users\night\repo\school\EGR491-PYMLDS>cd "PROJECT 1"
C:\Users\night\repo\school\EGR491-PYMLDS\PROJECT 1>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\night\repo\school\EGR491-PYMLDS\PROJECT 1>files
'files' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\night\repo\school\EGR491-PYMLDS\PROJECT 1>python 1_2_2_2-EX.py
High 5
C:\Users\night\repo\school\EGR491-PYMLDS\PROJECT 1>
```

2.1.1.3. Advanced Interpreter

There are other interpreters and IDE's for python. The best of these come in the conda package which includes the Spyder IDE and Jupyter Labs interpreter.

The difference between an IDE and Interpreter when it comes to python is that the IDE is more traditional. You write your code then you run it. In an interpreter you can write and run your code line by line or in short segments. Below is an example image from the Jupyter Labs extension for Visual Studio Code.



```
Interactive - #1 X
Jupyter Server: local Python 3.8.8 64-bit ('base': conda): Idle

Started 'Python 3.8.8 64-bit ('base': conda)' kernel
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.22.0 -- An enhanced Interactive Python. Type '?' for help.

[1] x = 100
    y = x / 2
    x *= y
    print(x, " ", y)

5000.0 50.0

[2] x -= 50
    print(x)

4950.0

[3] Type code here and press shift-enter to run
```

2.2. Syntax

Python has a very lax syntax compared to many languages. The worst or best of these is that python does not require a semicolon at the end of each line. But there are a few differences between python and languages like Java.

2.2.1. White Space

In Java white space is generally ignored and is simply used to increase readability. But python white space is used to distinguish blocks of code. For example the code below is invalid.

```
def sayHi():
print("hi")
```

And the output of the file makes it clear that line 2 is broken.

```
File "c:/Users/night/repo/school/EGR491-PYMLDS/examples/2-2_exmpl.py", line 2
print("HI")
^
```

To fix this simply add a couple spaces before your

```
def sayHi():  
    print("Hi")
```

Another attribute of python is that every function and variable's scope begins at its definition in the file or interpreter.

2.3. Variables

Python is an object oriented programming language and all variables available are objects. Unlike Java python variables are actually pointers. Because of this it does not matter the type of your variable. Consider the code below,

```
x = 5 # x is an integer  
x = 'BONJOUR' # x is a string
```

Those familiar with languages like Java and Ada will know that this does not happen normally. This is because of variables in python being pointers.

2.3.1. Built in Scalar Types

Python includes common built in types such as

- int
- float
- bool
- str (String type)

Python also includes a few uncommon types

- complex: Complex Number such as $1 + 1j$
- NoneType: This represents a Null value

2.3.2. Data Structures

Arrays are ubiquitous to programming and are necessary for many applications. Thus many languages have many types of arrays and other related data structures.

Python's four built in data structures cover many of the needs of data science and mathematics.

- **Lists** are common ordered lists and can be accessed by using a comma separated list with ","s between each item between square brackets.
- **Tuples** are immutable (unchanging) list. This is assigned list a normal list but with
- **Sets** These are unordered lists but sets can only have unique values. To assign a set put a comma separated list between curly brackets {}
- **Dicts** or dictionaries are sets with contain a key value mappings. to assign a dict to a variable each key value pair will be separated by commands with a colon between the key and value like this "x =

```
{'key1':1,'key2':2}"
```

Below is an example of how to use and access each of these arrays

2.3.2.1. Examples

```
lst = [1,"two",3, [4,5]]

print(lst)      # Prints the whole list
print(lst[0])   # Prints list item 1
print(lst[-1])  # Prints the last Item of the list
print(lst[2:])  # Prints Items from index to on
print(lst[:3])  # Prints Items 0 to three
print(lst[1:3]) # Prints Items 1, 2 and 3
print(lst[-3:]) # Prints the last three items
lst.append("THIS WORKS")
print(lst)
```

OUTPUT

```
[1, 'two', 3, [4, 5]]
1
[4, 5]
[3, [4, 5]]
[1, 'two', 3]
['two', 3]
['two', 3, [4, 5]]
[1, 'two', 3, [4, 5], 'THIS WORKS']
```

```
tup = (0,1,2)
# you can do the same things as you do with lists
print(tup)
print(tup[1])
# you cannot modify the tuple in any way
tup[1] = 7 # This gives a TypeError because
```

OUTPUT

```
(0, 1, 2)
1
Traceback (most recent call last):
  File "c:/Users/night/repo/school/EGR491-PYMLDS/PROJECT 1/Example3.py", line 6,
in <module>
    tup[1] = 7
TypeError: 'tuple' object does not support item assignment
```

```

setObj = {0,1,2}
#The Same as a tuple or a list
print(setObj)
# But you do have access to unique operations
print(setObj.union({2,3,4,5})) # Union: outputs all items contained in the sets
# duplicates are ignored you can also use |
print(setObj.intersection({1,7,9})) # Intersection: outputs the items shared in
# each set: you can also use &
print(setObj.difference({0,2})) # Difference: outputs the items in setObj but not
# in the internal set you can also use -
print(setObj.symmetric_difference({0,2,9})) # Symmetric Difference outputs items
# that are in either list but not both you can also use ^

```

OUTPUT

```

{0, 1, 2}
{0, 1, 2, 3, 4, 5}
{1}
{1}
{9, 1}

```

```

dic = {1:"THIS",2:"THAT",3:"NOT"}
print(dic[1]) # In this case you must access by a key not by the index
#Adding a new key value pair
dic["SWITCH"] = 99
print(dic)
print(dic["SWITCH"])

```

OUTPUT

```

THIS
{1: 'THIS', 2: 'THAT', 3: 'NOT', 'SWITCH': 99}
99

```

2.4. Math

In this solution we will show the basic mathematical operations of python. This includes the following:

Operation

+	Addition	Adds the two operators
---	----------	------------------------

Operation

-	Subtraction	Subtracts the first operator from the second
*	Multiplication	Multiplies the two operators
**	Power	Raser the first operator to the second
/	Nomal Dvision	Divides the frist operator by the second
//	Integer Division	Divides the first operator by the second with no remainder
%	Returns the integer remainder of a division between the first and second operator.	

- CODE

```
x = 7
print(x)
y = x + 5 # y = 12
print(y)
y = x * 2 # y = 14
print(y)
y = x ** 2 # y = 49
print(y)
y = x / 2 # y = 3.5
print(y)
y = x // 2 # y = 3
print(y)
```

- OUTPUT

```
7
12
14
49
3.5
3
```

2.5. Logic and Loops

2.5.1. Conditional Statments

Python conditional staments uses the following keywords

- *if*: opening statment for a logic block
- *elif*: the equivalent to else if in other languages
- *else* The final else stments

Python includes server bitwise operators that can be used in logic statments such as

&	AND
	OR
^	XOR
~	NOT

If statments are formed as follows

```
if x > 100:
    print("x is greater than 100")
elif x == 100:
    print("x is 100")
else:
    print("x is less than 100")
```

2.5.2. For loops

A python for loops looks like the code below,

```
for x in range(90, 100): #Prints all values from 90 to 100
    print(x)
```

There is also a specalse for of the for loop which includes an else statment which may be used when you use a break statment in a loop.

```
for x in [2,3,4]:
    if x == 7
        print(x)
        break
else:
    print("There is no 7")
```

This is not used as often as there are other ways to handle this type of operation.

2.5.3. While Loops

Below is a python while loop.

```
x = 10
while x > 0:
    if (x % 3) == 0:
        x *= 5
        x -= 7
        continue
    print(x)
    x -= 1
    if x > 100:
        break
```

You may notice that there is both a `break` and `continue` statement in this code block. The `continue` keyword essentially ends a iteration of a loop starting back at the beginning. The `break` keyword works just as it does in other languages.

2.6. Input and Output

A program is useless without a way to interface with a human. Python has many ways to interface with people and files but we will only cover a few basics here.

2.6.1. Screen

Python is often used in an interpreter and thus much of its input can come from there, but there are some times when you need to print to the console or receive input from it.

To print to the screen in python you simply use the `print()`

The print function can print strings and simple concatenation can work, but you can also input each item to print in a comma separated list.

```
#These two statements will print the same result
x = 100
print("x =" + x)
print("x =",x)
```

Input from the console is simply placing the `input()` function after an assignment operator.

```
print("Enter a value: ", end=" ")
x = input()
print("You Said ", x)
```

2.6.2. Files

When working with files you will need the `open()` function which has two parameters. The first is the file path, this can be either a relative path or an exact path. The second is the method to open the file with.

- "r" opens the file for reading if you assign this to a variable you can use .read() to read the whole file or .read(x) to read x number of lines.
- "a" will write to then end of a file with the .wrtie() function.
- "w" will write to the file and overwrite any exsisting content in the file.

Here is an example of how to read and write to files.

```
f = open("test.md", "w")
f.write("# This is a header\n")
f.close()
f = open("test.md", "a")
f.write("## This is a second level header\n")
f.close()
f = open("test.md", "r")

print(f.read())
```

OUTPUT

```
# This is a header
## This is a second level header
```

2.7. Functions

A function in python is white decalred using the **Def** keyword and its delaration must end with a colon.

```
#EXAMPLE FUNCTION
def myFunction():
    print("This is my function. There are many like it, but this one is mine.")
```

2.8. Imports

In Java you can get predesigned packages by doing, "import package_name."

In python there are not packages but **modules** which must first be installed. There are some good preinstalled such as numpy adds extended mathmatical abilities.

If you wish to use thrid party modules you follow the same syntax just ensure that they are installed. The most common way to instal python modules is using pip, below is an example of how to install the pygame modules which adds fun ways to make games.

```
pip install pygame
```

A warning to those who are just installing python for the first time. Make sure to install python to the PATH during the install process this will make using pip much easier.

There is also a way to import using the *from* keyword. This allows you to import specific componesnts of a modules of simply do "from <package> import *" to load all compoents of a modules.

3. Code Project

3.1. Sameple output

```
EGR491 Project 1: Paul A. Pace
This project is designed to display various simple attributes of python.
Please input either a comma separated list OR a file path.1,3;5,2,9
Reading List now!!!
1,3;5,2,9
INVALID VALUE:  3;5
The Sum of your list is  12
Your list without duplicates is  {1, 2, 9}
[2]
```

3.2. Code

```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'
# %%
# Function to check for a file
# returns TRUE if there is a file, else returns false
def check4File(filePath):
    import os.path
    return os.path.isfile(filePath)

# %%
# reads a file and loads it into a list, all new lines are treated as the default
separator
def readFile(path):
    f = open(path, "r")
    output = ""
    for l in f:
        output += ','
        output += str(l.strip())
    output = output[1:]
    return output

# %%
# Function that reads the users input and elminates all non recoverable values
```

```
def loadLst(strLst):
    lst = []
    for i in str(strLst).split(','):
        try:
            lst.append(int(i))
        except:
            print("INVALID VALUE: ", i)

    return lst

# %%
# Opening Title
print("EGR491 Project 1: Paul A. Pace")
print("This project is designed to display various simple attributes of python.")
userData = input("Please input either a comma separated list OR a file path.")

# %%
# Check for if userData is a file else read the list
if check4File(userData):
    print("Reading From your file")
    userData = readFile(userData)
else:
    print("Reading List now!!!")

# Print out userData
print(userData)

# %%
# Make your list from the userData and check for its sum
myList = loadLst(userData)

m = 0
for i in myList:
    m += i
print("The Sum of your list is ", m)
# %%

# Make a unordered list out of myList
myList2 = {i for i in myList}
print("Your list without duplicates is ", myList2)

# %%
# Make a list with only the even inputs
myList3 = [i for i in myList if i % 2 == 0]

# Print myList3
print(myList3)
```