Variables

Introduction to Data Science with Python

What are we Learning…

- This is a brief basic or crash course in Python. You won't be able to learn everything here, but enough to do the later exercises
- The goal is to give you a solid base, with sufficient knowledge to create basic Python programs and do some data science.
- I will give you links to other free/paid resources if you want to learn more about Python
- If anything of specific interest to you is not covered, let me know at the end of the session and I will attempt to briefly cover it at the next opportunity

Multiple Commands on One Line

- You do not need to end lines with any special characters
- However, you can put multiple commands on one line by separating them with a semi-colon
- I don't often do this, I don't mind long files with short widths
  - They tend to read better

```
# Multiple Commands on One Line
print("Command 1") ; print("Command 2")
>>Command 1
>>Command 2
```

Strong vs Weak Typing

- Python is considered a “weakly” typed language

  Since it is a scripting language, variables are typed at runtime, unlike compiled languages

  This means you don’t have to declare a variable type before using it

  Python determines the type from the assignment

```
# Character type
var1 = "foo“
# Numeric type
var1 = 7
```

String Literal

- You will spend a lot of time working with strings
- Use single or double quotes (matching) to define a string
- You can use triple quotes for docstring or multiline string

```
# Strings
var1 = "This is a string."
var2 = ‘This is also a string.’
var3 = """This is a
multiline string
"""
print(var1)
This is a string.
print(var3)
This is a
multiline string
```

String Literals

- String are Immutable (they can't be altered)
- Use str() function to convert a variable to a string equivalent
- Strings are sequences of characters, and therefore iterable

```
# A string is immutable so this throws an error
var1[2] = "c"
# Type conversion using str
x = 10
var5 = str(x)
# Iterate over a string
for i in var1:
    print(i)
```

String Literals

- Indexing strings
- Escape character \ (backslash)
  - Must use double escape to get the actual \ character
  - Can preface string with r"…" to indicate raw string which will ignore escape characters

```
# Index
var1[0:3]
>> 'Thi‘

# Windows Path Example
print("C:\Data\project")
print("C:\Data\b1")
print(r"C:\Data\b1")
```

Escape Characters

Character Sequence	String Result
\\	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	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)

From Official Python Docs

String Formatting

- You can put a “place holder” for strings, then pass in arguments.

  The .format() method. We can also format the argument as we call it. See the python documentation for further details

  https://docs.python.org/3/library/string.html#string-formatting

```
# String Formats
print("Hello, my name is {name}".format(name="Rufus"))
print("Hello, my name is {fname}
      {lname}".format(fname="Rufus",lname="McGee"))
print("Hello, my name is {0}     {1}".format("Rufus","McGee"))
print("Hello, my name is {} {}".format("Rufus","McGee"))
print("Hello, my name is {0:_<5}
      {1}".format("Rufus","McGee"))
```

Capturing User Input

- We can prompt a user for input then use it in the program

  Example - **input**("Prompt")

```
# User Input
player1 = input("What is your name, player? ")
print("Welcome, ", player1)


# Square Number
num1 = input("Enter the number you would like me
to square: ")
num2 = float(num1)
print("The result is ", num2 * num2)
```

Test Your Knowledge – Ex 3.1

- Now it's your turn to apply what you have learned.

```
Write Python code to do the following:

1. Prompt for input "Your first name is: "

2. Prompt for input "Your last name is: "

3. Output "Your name is: " and string together your first and last
name separated by a space.
```

Numbers

- We will consider integers and floats
  - Large integers will be converted to “long integer”
  - Floats are double precision (64 bit)
  - Division returns integers in Python <= 2.7x

```
# numbers
var1 = 43
var2 = 9
var1 / var2
## Python 2.xx behavior
4
float(x) / y
4.77777778
# python 3.x behavior
x / y
4.77777778
```

math Module

- Python provides the math module for numeric functions
  - Log
  - Sqrt
  - Lots of others

```
import math

var1 = 10

math.sqrt(var1)
```

Numeric Type Casting

- Use int() or float()

  Characters or internal whitespace will cause an error. Leading and trailing whitespace with numbers inside will convert.

```
# Type Conversion
int("  45")
45
float("4.769  ")
4.769
int("b6")
ValueError: invalid literal for int()...
float("4.68 97")
ValueError: invalid literal for float()
```

Boolean

- True and False
  - Result of comparisons and many other expressions.
  - Combine with and / or
  - Most python classes have a boolean resolution (useful in loops)
  - Use the bool() function to find the resolution

```
# Boolean
print("x =",x,", y=", y)
>> x = 43 , y= 9
x < 50
x < 50 and y > 15
x < 50 or y > 15
if x:
    print("x is not empty!“)

bool(x)
```

Null Values

- None is the base Python reserved null value
- We will see more null value types introduced with subsequent packages

```
# Null
var1 = 43
var2 = None
print("not empty" if var1 else  "empty!“)
print("not empty" if var2 else  "empty!“)
bool(var2)
```

Lab 3 (20 min)

- Write a Python script to do the following:

1. Prompt the user to input a number
2. Output the log of that number (base 10)
3. Output the nearest integer less than that number

4. Prompt the user to input two numbers
5. Add those two numbers and print the result as a string: ‘The sum of those two numbers is xx’