COMP2152 LAB MANUAL

OPEN SOURCE DEVELOPMENT

This booklet will help the reader understand the concepts, principles, and implementation of the Python programming language. By the end of the booklet, the reader will be able to code comfortably in Python.

Prepared by Ben Blanc

TABLE OF CONTENTS

CHAPTER 1

GETTING STARTED

REQUIRED TOOLS

There is one application and one interpreter that needs to be downloaded on your personal machines to complete the lab manuals and practice coding.

PyCharm

Following the instructions and video given to you in Week 1 on GBLearn. The instructions include

- If you don't have a JetBrains account, signing up for a Student Account with JetBrains at https://www.jetbrains.com/shop/eform/students
    - Use your George Brown email to signup
    - Confirming the confirmation email received from JetBrains.
- Download the PyCharm product
- Downloading the JetBrains license
- Pasting the license onto your personal machine. (You may also validate the license by using your JetBrains account credentials)

Python is downloaded automatically when you download PyCharm.

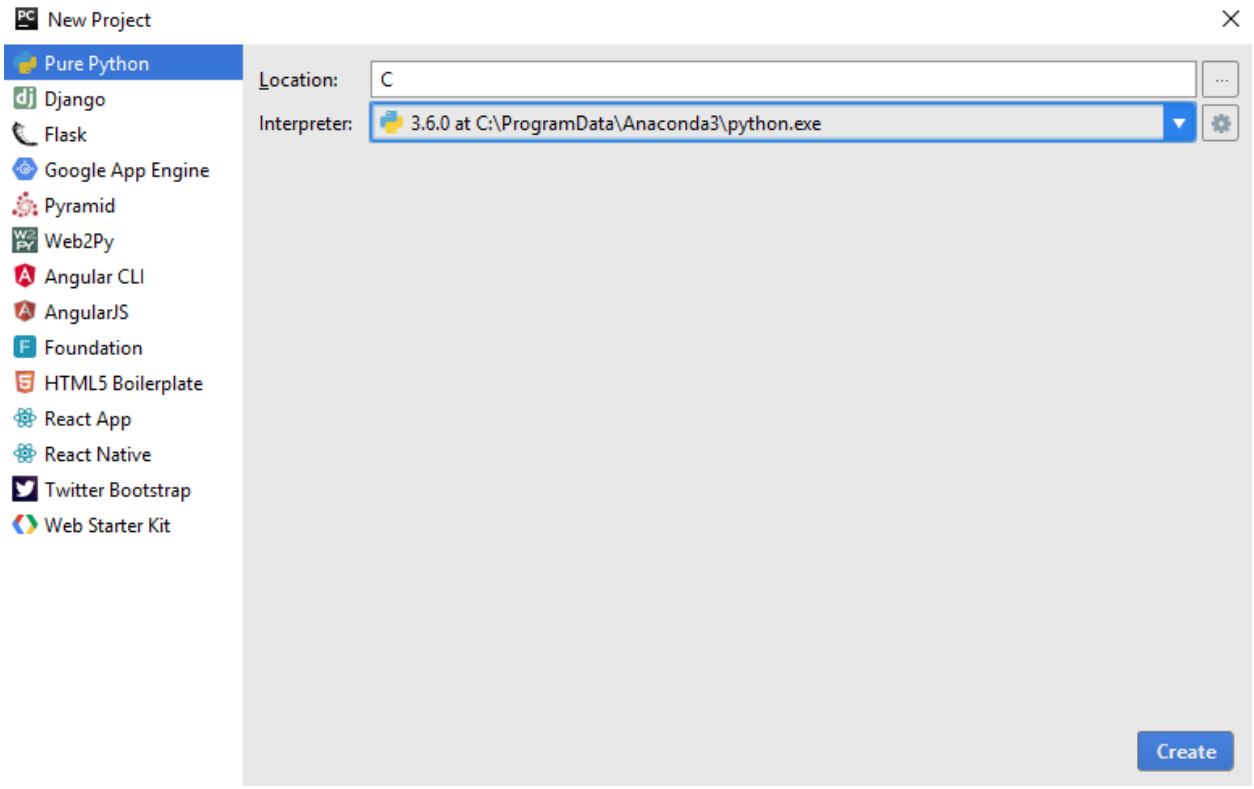
CHAPTER 1 AT A GLANCE

In this chapter you will learn where to download and how to install the required tools to run a Python script. You will also learn how to

- output information
- receive user input
- test and debug
- work with data types
- work with variables
- work with number data type

CREATING FIRST PYTHON APP

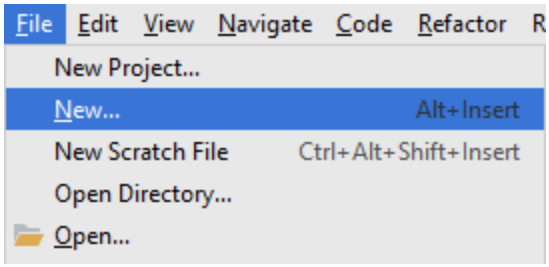
Open PyCharm storm and click on New Project



Choose the location of the project and select the Python Interpreter.

For the interpreter, click on the dots, click on the plus sign on the top left corner of the new window (Other local), and navigate to the location of your python interpreter location. After completing that step, you can select a Python interpreter from the drop down

Once you have finished, you will see a blank project. Click on File -> New. Then select Python file and fill out a name. Our filename will be main.py. This file is automatically created for you in your project.



Delete all of the default text in main.py. Add the following code to the page:

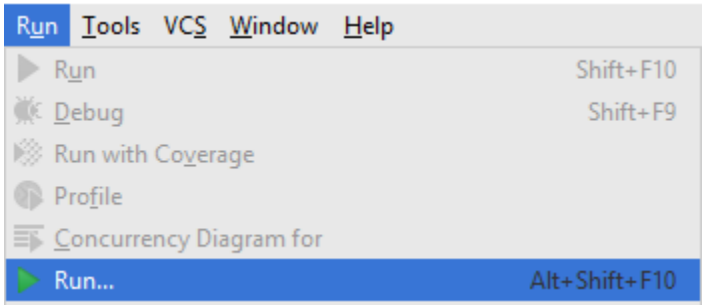
```
print("Hello World")
```

RUNNING YOUR PYTHON APP

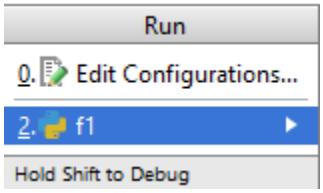
Now it is time to run your first python application.

Running PyCharm's Python Source Code

Click on Run->Run to execute your Python source code



There will be another pop-up to run your specified file



Click on the filename and the file will run.

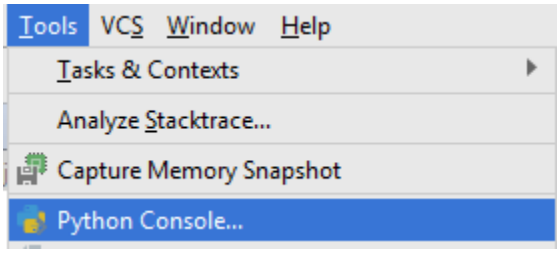
```
Hello World

Process finished with exit code 0
```

When you see the message "Process finished with exit code 0", you know your code has finished executing.

In PyCharm's Interactive Mode

Click Tools -> Python Console…



This will open the Python Console that you can run Python code

Typing In **print("Hello World")** in the console and pressing [enter] will result in the following:

Use the Interactive Mode to run test code that you don't want to necessarily have in your source code. All variables that you declare will persist until you close the session.

WORKING WITH DATA TYPES AND VARIABLES

We will cover four of the many data types in python.

Declaring a Variable

To declare a variable in python, state the identifier name, an equal sign, and its value. Variables in Python are case-sensitive

Variable_Name = Value

PYTHON VARIABLE RULES

- Must begin with an alphabetical letter or an underscore (_)
- Other characters can be alphanumerical and underscore
- Cannot use python reserved words

Examples of valid python identifiers

```
salut = "Mister"

number = 123

right = True
```

STRING

To output a string, surround the value with single or double quotes.

```
print("A double quoted string value")

print('A single quoted string value')
```

You can also put string values into a variable and output the variable

```
str1  = "A double quoted string value"

str2 = 'A single quoted string value'

print(str1)

print(str2)
```

INTEGER

To output an integer, output a non-decimal the value without quotes.

```
print(123)

print(987)
```

You can also put integer values into a variable and output the variable

```
num1 = 456

num2 = 279

print(num1)

print(num2)
```

FLOAT

To output a float, output a decimal the value without quotes.

```
print(12.34)

print(56.78)
```

You can also put float values into a variable and output the variable

```
fl_1 = 544.363

fl_2 = 793.42

print(fl_1)

print(fl_2)
```

BOOLEAN

To output a Boolean, output the values True or False without quotes.

```
print(True)

print(False)
```

You can also put Boolean values into a variable and output the variable

```
bool_1 = True

bool_2 = False

print(bool_1)

print(bool_2)
```

WORKING WITH NUMERIC DATA TYPE

You can work with numbers in similar ways as other programming languages.

Arithmetic Operators

The following are supported arithmetic operators in Python

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Integer Division (gets floor of result of a / b)
%	Modulus (remainder of quotient operation)
**	Exponentiation

```
print(1+5)

print(10-3)

print(6*3)

print(9*5)

print(12/5)

print(12//5)

print(20%3)

print(2**4)
```

```
num1 = 52
num2 = 5

print(num1 + num2)

print(num1 - num2)

print(num1 * num2)

print(num1 / num2)

print(num1 // num2)

print(num1 % num2)

print(num1 ** num2)
```

Compound Assignment Operators

Compound assignment operators is a shorter syntax for updating a variable. All of arithmetic operators can be used in shorthand The following are supported compound assignments:

- **+=**
- **-=**
- **/=**
- **//=**
- ***=**
- **%=**
- ****=**

```
num1 = 12

num2 = 4

num1+=7

print(num1)

num1-=num2

print(num1)

num2*=2

print(num2)

num2*=num1

print(num2)
```

Order of Precedence

The order of presence is demonstrated by the following table. Operations in brackets have the highest priority.

Order	Operator	Direction
1	**	Left to right
2	* / // %	Left to right
3	+ -	Left to right

Your turn! Describe the order of the second mathematic statement:

```
e+= (( a + b) * c ) / d
```

```
a = -12

b = 15

c = 130

d = 75

e = (a + b) * c / d

e+= (( a + b) * c ) / d
```

For the first e value, the order of operations is as follows

- First the brackets statements are evaluated. Statements in brackets are always evaluated first.
  - Since there is only one statement, the addition operation is executed
- Upon exiting the bracket,
  - The next operation is the multiplication of (a+b) * ++c
  - Lastly, the division of the above statement by d is performed

CONCATENATION & SPLITTING STATEMENTS

Concatenation

Concatenation is merging two values together. In python, the two values must be of the same data type. The concatenation symbol is the plus character: **+**

```
print("Hello" + " " + "World")

str1 = "Hello"

str2 = "World"

num1 = 5

print(str1 + " " + str2)

print("Hello" + 1)

print(str1 + num1)
```

The last two print statements produce errors

Splitting Statements

You can split python statements by the following characters

- +
- \

```
print("Hello "\
      "There " \
      "From " \
      "Multiple " \
      "Lines")


print("Hello "+
      "There " +
      "From " +
      "Multiple " +
      "Lines")
```

ESCAPE CHARACTERS & COMMENTS

In order to display tabs, new lines and special characters that have programming meanings, they need to be escaped.

```
print("\n")
print("\r")
print("\r")
print("\'")
print("\"")
print("\\")
```

Character	Description
\n	Displays a new line
\t	Displays a tab indentation
\r	Displays the return line character
\’	Displays a single quote
\”	Displays a double quote
\\	Displays a backslash

To create a comment in Python, use the hashtag symbol (#)

```
#this is a comment
```

COMMON PYTHON FUNCTIONS

print()

print() is used to output data. You can also output multiple arguments:

```
print("Hello", "World", "From", "Python")
```

Multiple arguments can be any data type. By default, the arguments are separated by a single space.

```
print("Hello", True, 468, 98.76)
```

You can change the separator of the arguments by the following code:

```
print("Hello", "World", "From", "Python", sep='\t')

print("Hello", True, 468, 98.76, sep='\t')
```

In the above statements, the separator has been changed to the tab character.

You can also change the ending character of a print statement from the default value of new line to another string.

```
print("Hello", "World", "From", "Python", end='--END--')

print("Hello", True, 468, 98.76, end='--END--')
```

You can add the separator and ending character in one print statement

```
print("Hello", "World", "From", "Python", sep='\t', end='--END--')

print("Hello", True, 468, 98.76, sep='\t', end='--END--')
```

input()

input() is used to get input from a user. Optionally, you can pass a message to the user.

```
input()

input("Eenter your name")

input("Enter number")
```

A string data type is returned. To store the input, create an identifier to store the user value.

```
user_input = input()

name = input("Eenter your name")

number = input("Enter number")
```

str(), int(), float()

These three functions typecast/parse the values to the specified data types.

```
str_from_number = str(1)

str_from_boolean = str(True)

str_from_float = str(12.34)

float_from_int = float(2)

float_from_str = float("56.78")

float_from_boolean = float(False)

int_from_boolean = int(False)

int_from_float = int(123.45)

int_from_str_1 = int("123")

int_from_str_2 = int("123.45") #throws value error
```

round()

round() rounds a number value to the specified precision. It takes a number data type as the first argument and optionally, a precision.

```
print ( round(123.45) )

print ( round(123.45, 1 ) )
```

len()

len() determines the number of items in a collection data type (i.e., string, list, tuple, dictionary). It takes one required argument.

```
a = "Hello World"

print(len(a))
```

FORMAT OUTPUT

You can format the output in python by adding padding before or after the output. You can also represent data as various data types.

To format an output or string

- Add braces
- optionally, add an argument number or argument name
- add a colon to describe the data type
- optionally, add a width specifier that the data value will occupy
- add a format character
- optionally, add a precision number

Below is sample code with formatted output.

```
print("The price is ${0:7.2f}.".format(19.9) )
```

Output will be

```
The price is $  19.90.
```

Here is an explanation of the code above:

- The **0** represents the argument number
- The **7** represents the width that the data value will occupy
- The **.2** represents the precision of that data value
- The **f** represents the data type (float)

Here are other examples that have the exact same output as above

```
print("The price is ${:7.2f}.".format(19.9) )

print("The price is ${val:7.2f}.".format(val=19.9) )
```

Here are more examples of formatted strings

```
shoes = 2
socks = 1.5
str1 = "I have {:d} pairs of shoes and " \
       "{:f} pairs of socks".format(shoes, socks)

str2 = "I have {0:d} pairs of shoes and " \
       "{1:f} pairs of socks".format(shoes, socks)

str3 = "I have {numShoes:d} pairs of shoes and " \
       "{numSocks:f} pairs of socks" \
        .format(numShoes=shoes, numSocks=socks)

str4 = "I have {0:d} pairs of shoes and " \
       "{1:.1f} pairs of socks".format(shoes, socks)

str5 = "I have {numShoes:d} pairs of shoes and " \
       "{numSocks:.1f} pairs of socks"\
        .format(numShoes=shoes, numSocks=socks)

str6 = "I have {:1d} pairs of shoes and " \
       "{:8f} pairs of socks".format(shoes, socks)

str7 = "I have {0:1d} pairs of shoes and " \
       "{1:8f} pairs of socks".format(shoes, socks)

str8 = "I have {numShoes:1d} pairs of shoes and " \
       "{numSocks:8f} pairs of socks"\
        .format(numShoes=shoes, numSocks=socks)

str9 = "I have {0:1d} pairs of shoes and " \
       "{1:8.1f} pairs of socks".format(shoes, socks)

str10 = "I have {numShoes:1d} pairs of shoes and " \
        "{numSocks:8.1f} pairs of socks"\



print(str1)
print(str2)
print(str3)
print(str4)
print(str5)
print()
print(str6)
print(str7)
print(str8)
print(str9)
print(str10)
```

Output will be

```
I have 2 pairs of shoes and 1.500000 pairs of socks
I have 2 pairs of shoes and 1.500000 pairs of socks
I have 2 pairs of shoes and 1.500000 pairs of socks
I have 2 pairs of shoes and 1.5 pairs of socks
I have 2 pairs of shoes and 1.5 pairs of socks

I have 2 pairs of shoes and 1.500000 pairs of socks
I have 2 pairs of shoes and 1.500000 pairs of socks
I have 2 pairs of shoes and 1.500000 pairs of socks
I have 2 pairs of shoes and      1.5 pairs of socks
I have 2 pairs of shoes and      1.5 pairs of socks
```

WIDTH SPECIFIER

You can specify width spacing when formatting output by using the **ljust()**, **rjust()** or **center()** methods.

The **ljust()** method aligns the text to the left-hand side, adding the padding to the right hand side.

The **rjust()** methodaligns the text to the right-hand side, adding the padding to the left hand side.

The **center()** method centers the text evenly.

To add spacing before an output, use the **rjust()** method

```
print("Left Spacing".rjust(15))
```

Produces the following result

```
   Left Spacing
```

You can an a secondary parameter to specify what the filler character is

```
print("Left Spacing".rjust(15,'*'))
```

Produces the following result

```
***Left Spacing
```

The following code:

```
print( "Left Spacing".rjust(15), "Middle".ljust(10), "Right Spacing".center(17) )

print( "Left Spacing".rjust(15,'#'), "Middle".ljust(10,'|'), "Right Spacing".center(17,'*') )
```

Produces the following result

```
   Left Spacing Middle       Right Spacing
###Left Spacing Middle|||| **Right Spacing**
```

ERRORS

The first kinds of errors you may encounter in Python are syntax, type, name and division by zero error

SyntaxError

A syntax error is an error that does not allow your application to compile because it violates a rule of the language.

An example of a syntax error is mis-matching quotes.

```
print('")
```

TypeError

This error occurs when you try to complete an operation on a data type in which that operation is not supported.

```
print("1" - 1)

a = 12 / "3"
```

For both statements, you are trying to perform arithmetic operations on non-numerical values. Also, when you attempt to concatenate, the data types need to be the same.

NameError

This error happens when you attempt to refer to an identifier/variable that has yet to be declared

ZeroDivisionError

This error happens with you attempt to execute a division statement with 0 as the denominator.