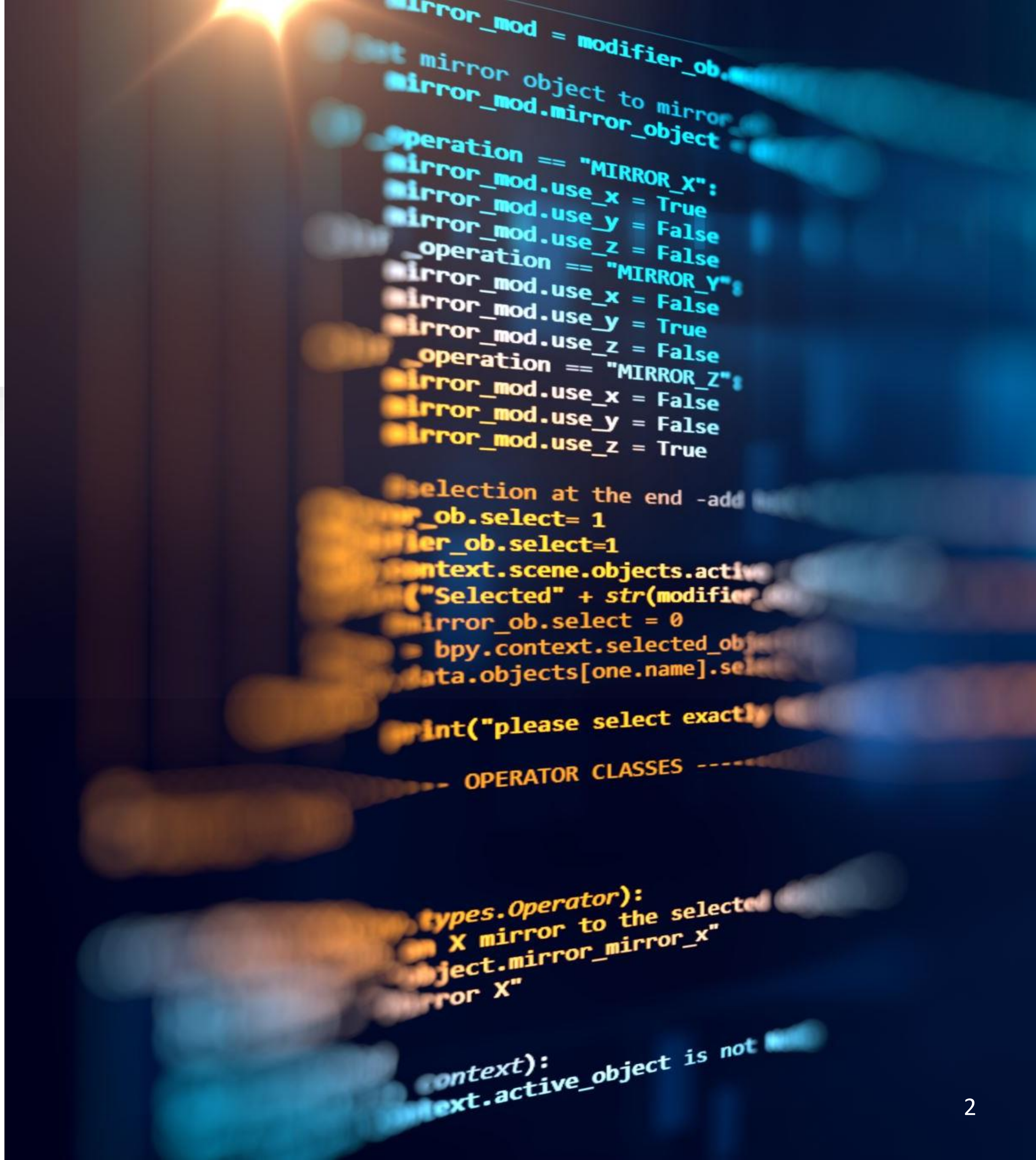
COMP2853

LESSON 1: SETUP, VARIABLES, EXPRESSIONS, NAMING

Agenda – Lesson 1

- Course Logistics
- Development Toolset
- First Python Program
- Programming and Python Basics
- Lab 1



Instructor

- Instructor: Johnny Zhang, MS, Ph.D.
- Interests:
  - Cloud Computing Architecture and Design
  - Programming – Python, JavaScript, C, C++
  - Software Project Management
  - Network Security
  - Windows / Linux Server
- Office Hours:
  - By Appointment
- E-mail: johnny_zhang@bcit.ca (subject: COMP2853)

Students

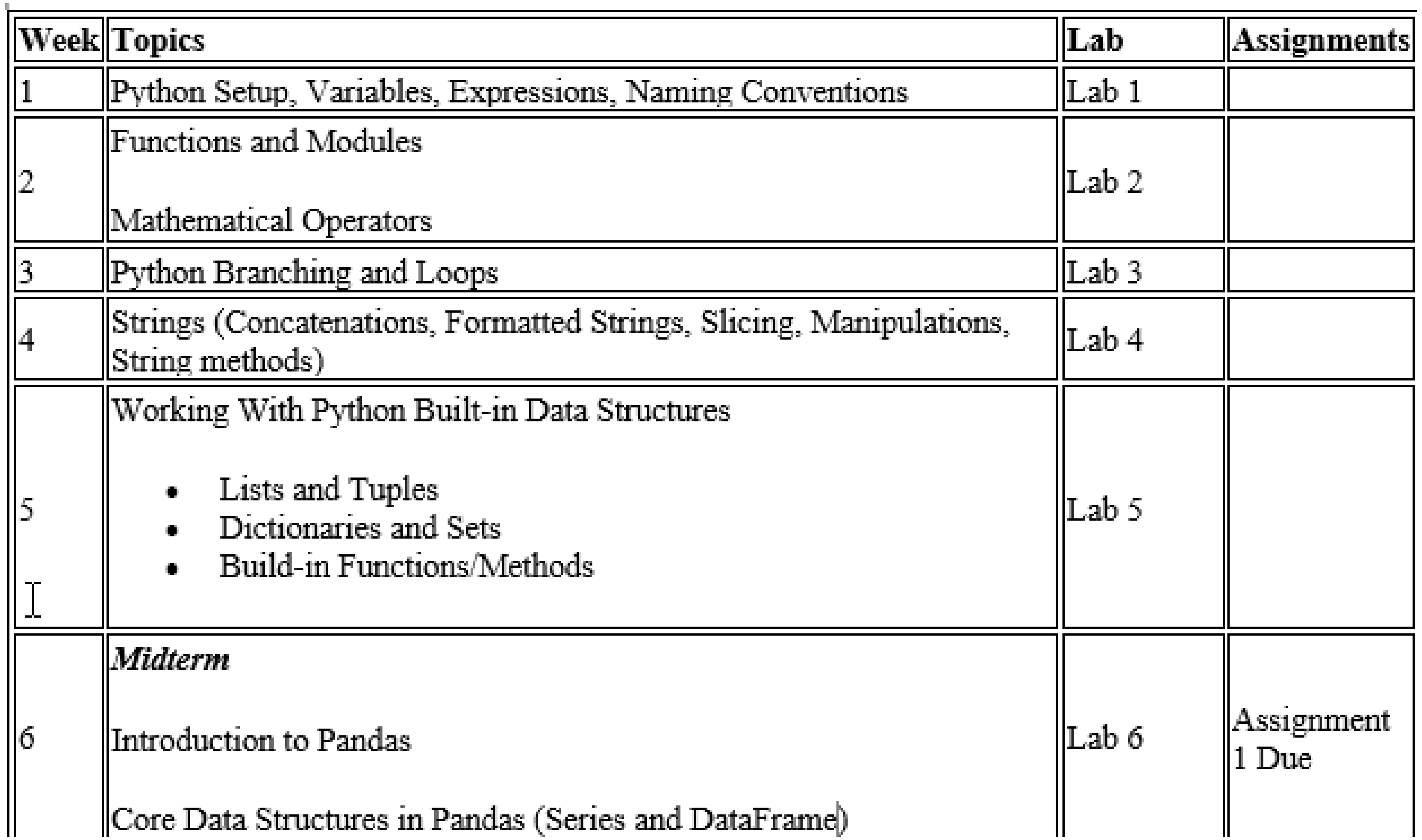
- Submit your homework to learn.bcit.ca -> COMP 2853 -> Activities -> Assignments
- Check email regularly at my.bcit.ca and course news on learn.bcit.ca
- Ask questions
- Email attachment (notepad)
- Contact your instructor by email (johnny_zhang@bcit.ca)
- Do not need any programming experience

Please introduce yourself to the class and tell us why you're taking this course.

Course Delivery

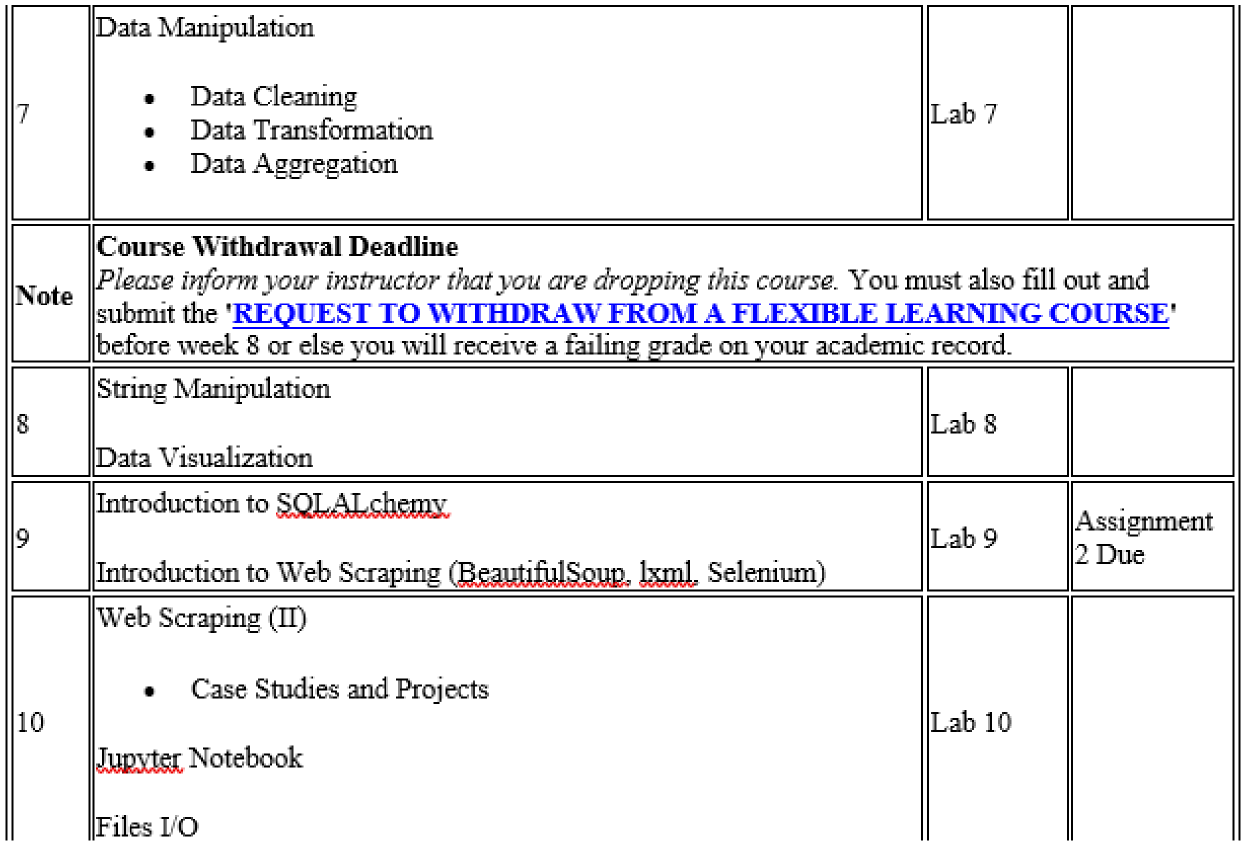
- Course Site on BCIT’s Learning Hub
  - New Items
  - Virtual Classroom (Online Session or Weather Conditions Only)
  - Submit Labs and Assignments
  - Quizzes
  - Midterm && Final Exam
- In-Person classes weekly from 9-12 on Saturday Morning
  - Includes Quiz, Lecture, and Lab.
- PDF slides will be posted to the Course Site
- Quizzes will be held at the beginning of each class
- Homework will be submitted every week

Course Outline

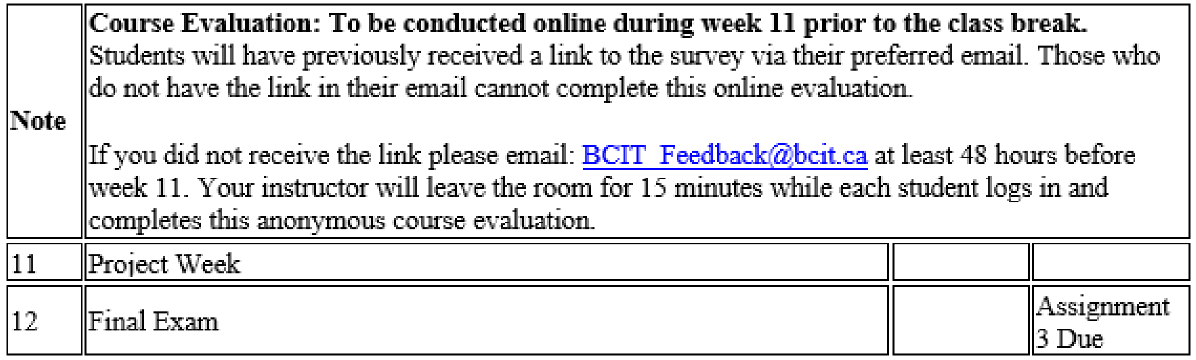


Week	Topics	Lab	Assignments
1	Python Setup, Variables, Expressions, Naming Conventions	Lab 1	
2	Functions and Modules Mathematical Operators	Lab 2	
3	Python Branching and Loops	Lab 3	
4	Strings (Concatenations, Formatted Strings, Slicing, Manipulations, String methods)	Lab 4	
5	Working With Python Built-in Data Structures • Lists and Tuples • Dictionaries and Sets • Build-in Functions/Methods	Lab 5	
6	***Midterm*** Introduction to Pandas Core Data Structures in Pandas (Series and DataFrame)	Lab 6	Assignment 1 Due

Course Outline

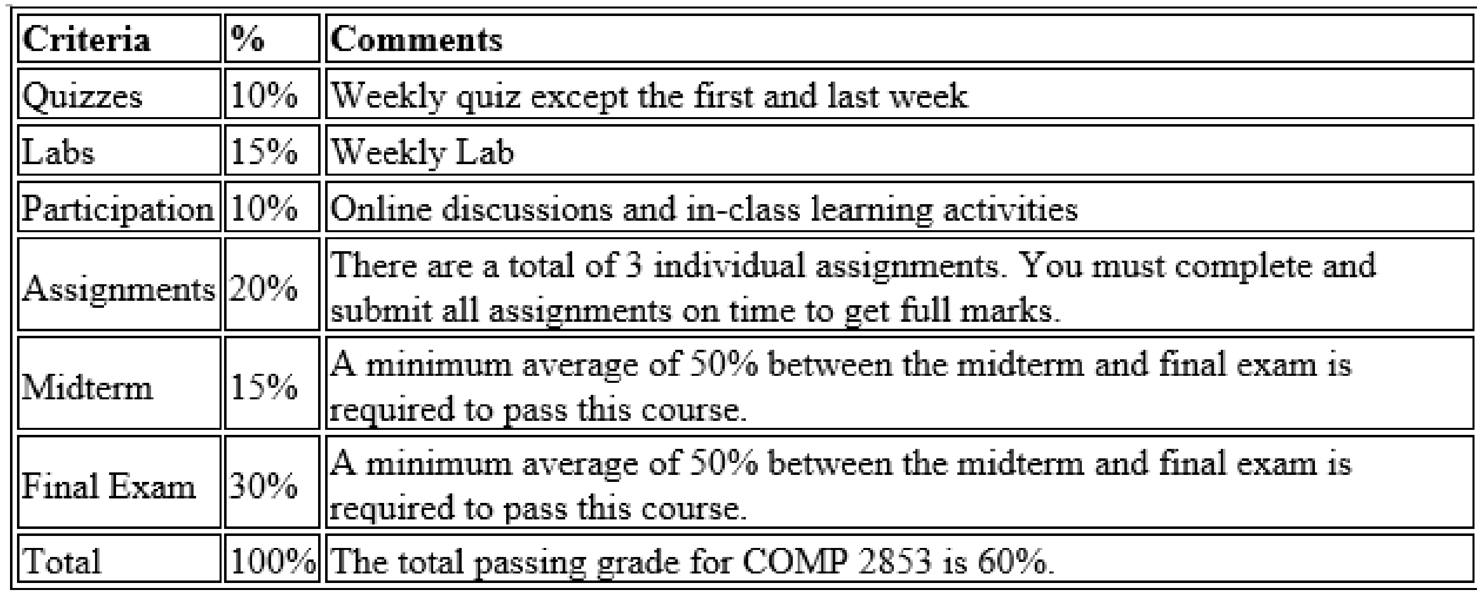


7	Data Manipulation • Data Cleaning • Data Transformation • Data Aggregation	Lab 7	
Note	**Course Withdrawal Deadline** *Please inform your instructor that you are dropping this course.* You must also fill out and submit the '**REQUEST TO WITHDRAW FROM A FLEXIBLE LEARNING COURSE**' before week 8 or else you will receive a failing grade on your academic record.		
8	String Manipulation Data Visualization	Lab 8	
9	Introduction to SQLALchemy Introduction to Web Scraping (BeautifulSoup, lxml, Selenium)	Lab 9	Assignment 2 Due
10	Web Scraping (II) • Case Studies and Projects Jupyter Notebook Files I/O	Lab 10	



Note	**Course Evaluation: To be conducted online during week 11 prior to the class break.** Students will have previously received a link to the survey via their preferred email. Those who do not have the link in their email cannot complete this online evaluation. If you did not receive the link please email: BCIT_Feedback@bcit.ca at least 48 hours before week 11. Your instructor will leave the room for 15 minutes while each student logs in and completes this anonymous course evaluation.		
11	Project Week		
12	Final Exam		Assignment 3 Due

Evaluation Criteria



Criteria	**%**	**Comments**
Quizzes	10%	Weekly quiz except the first and last week
Labs	15%	Weekly Lab
Participation	10%	Online discussions and in-class learning activities
Assignments	20%	There are a total of 3 individual assignments. You must complete and submit all assignments on time to get full marks.
Midterm	15%	A minimum average of 50% between the midterm and final exam is required to pass this course.
Final Exam	30%	A minimum average of 50% between the midterm and final exam is required to pass this course.
Total	100%	The total passing grade for COMP 2853 is 60%.

- Students must average a minimum of **50% between the Midterm** and the **Final Exam** to pass this course.
- Students must attempt and submit each lab and assignment to pass the course.
- Students must get at least 60% in the course overall to pass.

Class Structure

There is a mandatory online reading/exercise/discussions that must completed before class

Before Class:

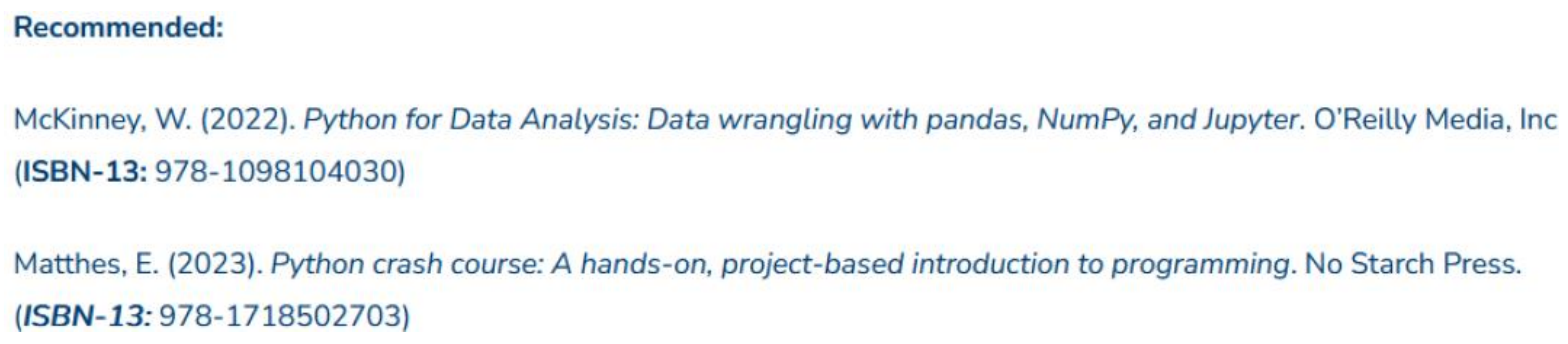
- Complete the at home reading/exercise posted to the Course Site
- Email your questions/comments directly to me

General Class Agenda:

- Quick review of previous week's topics and challenge questions / practices
- Quiz
- Lecture and Coding Demos
- Lab

Learning Resources

- No textbook is required for this course.



Recommended:

McKinney, W. (2022). *Python for Data Analysis: Data wrangling with pandas, NumPy, and Jupyter*. O'Reilly Media, Inc (**ISBN-13:** 978-1098104030)

Matthes, E. (2023). *Python crash course: A hands-on, project-based introduction to programming*. No Starch Press. (***ISBN-13:*** 978-1718502703)

- Extra materials: www.py4e.com (it has video lectures, slides and an online book chapter. The materials are free to use.)

Data Analysis

Data Analysis is a process of inspecting, cleaning, transforming, and modeling data with the purpose of discovering useful information, suggesting conclusions, and supporting decision-making.

Key Stages of Data Analysis:

- Data Collection
- Data Cleaning
- Data Exploration
- Data Transformation
- Data Modeling
- Interpretation
- Visualization
- Reporting

What is Programming?

Programming is the process of designing and creating instructions for a computer to follow.

Steps in the Programming Process:

- Understand the problem (Clarify the requirements and break down the problem)
- Design a solution (Design an Algorithm, Choose the right data structure(array, list, dic..))
- Select the programming language (i.e., Python)
- Write the code (Work on your code in an IDE)
- Test the code
- Debug the code (i.e., troubleshoot problems)
- Optimize the code (improve performance, refactor)
- Document the ode (add comments, write documentation)
- Maintain the code (Bug fixes and updates)

Some Programming Languages

Compiled Languages

Converted to Native Computer Instructions

Scripting Languages

Run in an interpreter program

What might be the advantages of a compiled language?

What might be the advantages of a scripting language?

Python

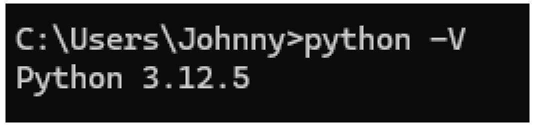
- This course only focuses “ Python basic programming fundamentals”… not an in-depth course in Python.
- But we will use Python to learn and apply our knowledge.
- Python is an interpreted language, like JavaScript.
- It is not a compiled language, like Java or C++.

More Notes on Programming

- Much of the time spent in developing programs happens in two areas:
  - Understanding the problem, you are trying to solve
  - Fixing bugs in your software
- It is important you ask questions to clarify when you need help
- Your code likely won't work the first time – you'll need to debug your code

Course Tools

- Python 3 (Latest – 3.10 or later)
    - Download: https://www.python.org/downloads/
- PyCharm IDE (2024 or later)
    - Free Student License: https://www.jetbrains.com/student/
    - Note: Community Edition is okay too



```
C:\Users\Johnny>python -V
Python 3.12.5
```

Our First Python Program

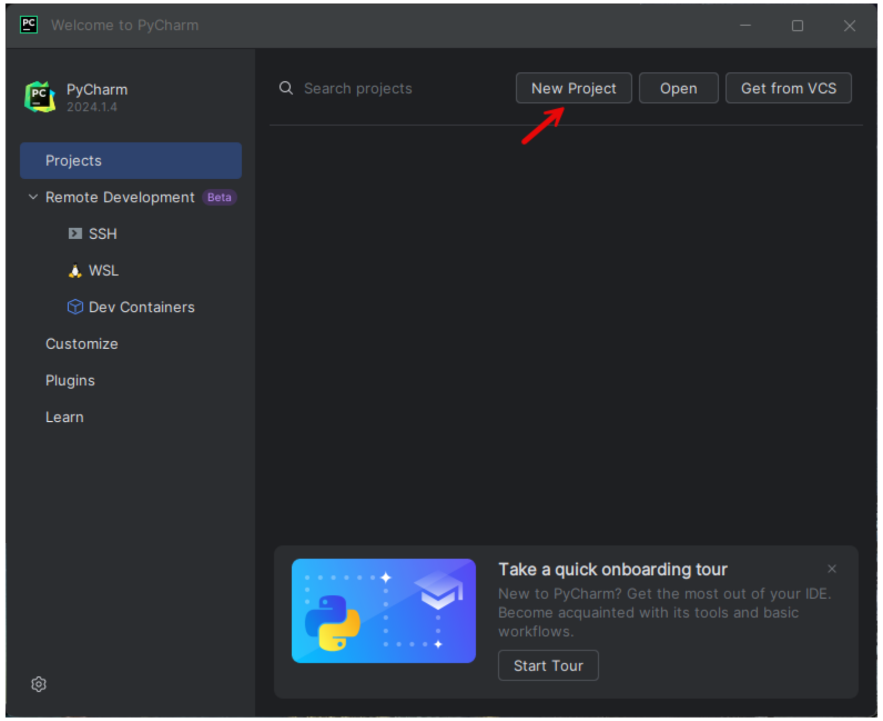
Create a Project

File

New Project

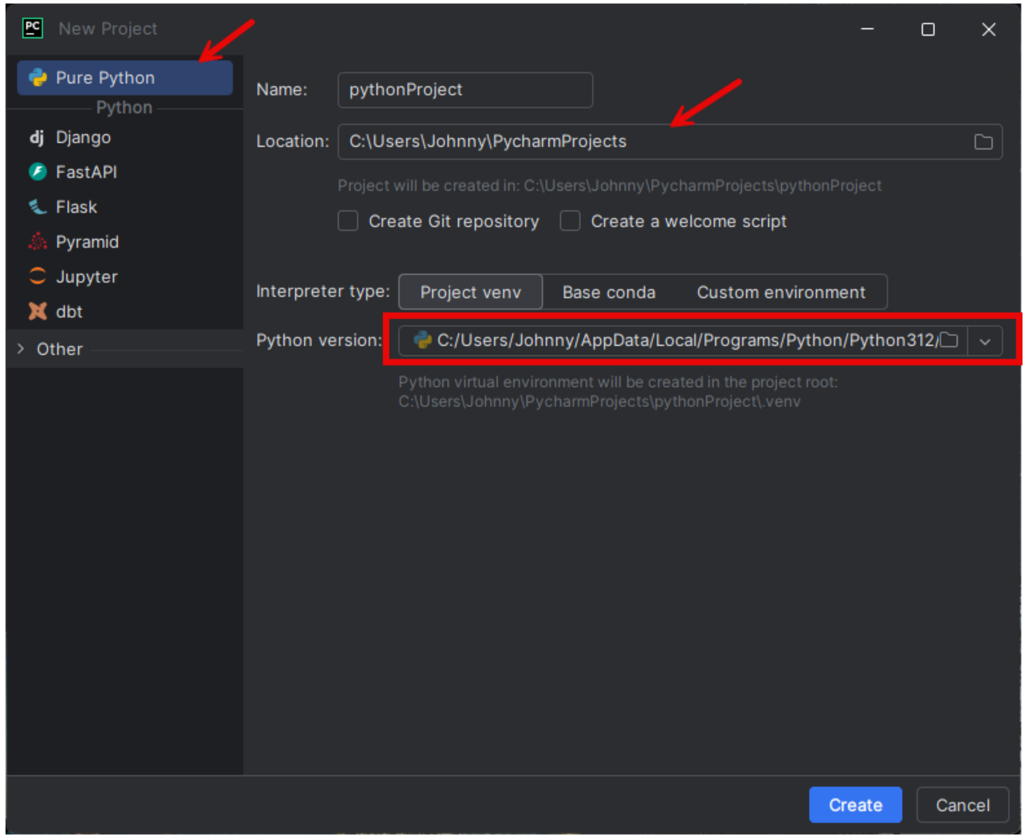
Note:

May be slightly different on a Mac



Take a quick onboarding tour

New to PyCharm? Get the most out of your IDE. Become acquainted with its tools and basic workflows.



Name: pythonProject

Location: C:\Users\Johnny\PycharmProjects

Project will be created in: C:\Users\Johnny\PycharmProjects\pythonProject

Create Git repository | Create a welcome script

Interpreter type: Project venv | Base conda | Custom environment

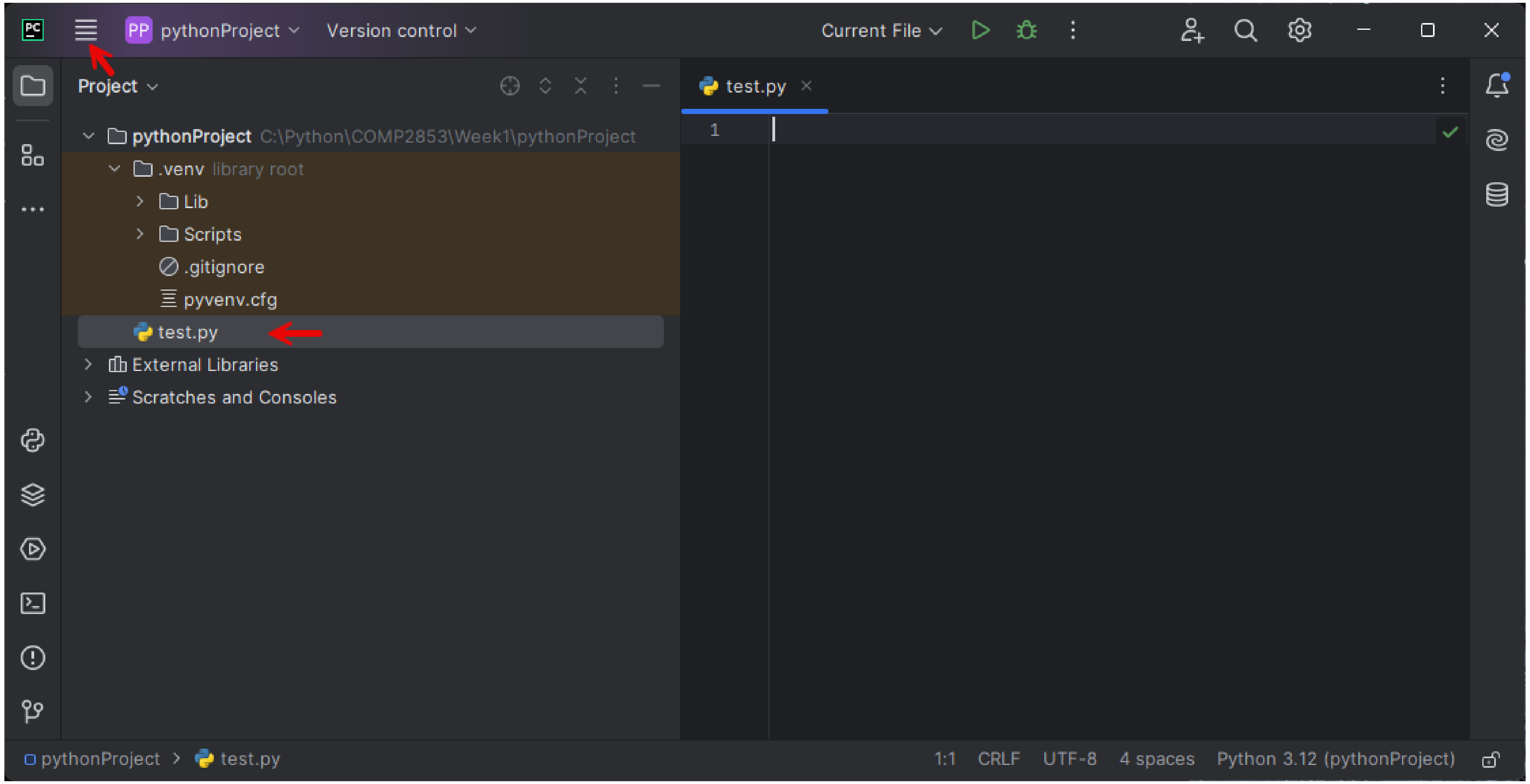
Python version: C:/Users/Johnny/AppData/Local/Programs/Python/Python312/

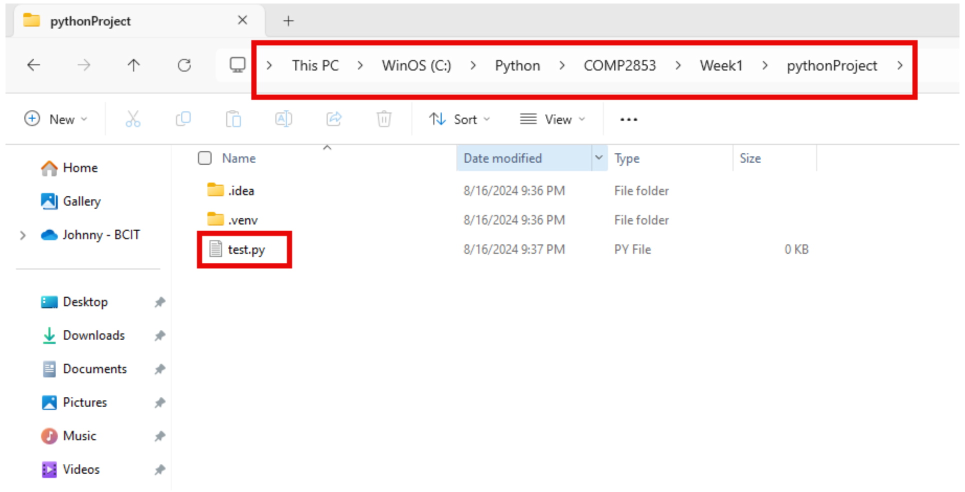
Python virtual environment will be created in the project root: C:\Users\Johnny\PycharmProjects\pythonProject\.venv

Our First Python Program

- File
- New…
- Python File
- Enter "test"

- Notice "test.py"
- **Run**





This PC > WinOS (C:) > Python > COMP2853 > Week1 > pythonProject >

Name	Date modified	Type	Size
.idea	8/16/2024 9:36 PM	File folder	
.venv	8/16/2024 9:36 PM	File folder	
test.py	8/16/2024 9:37 PM	PY File	0 KB

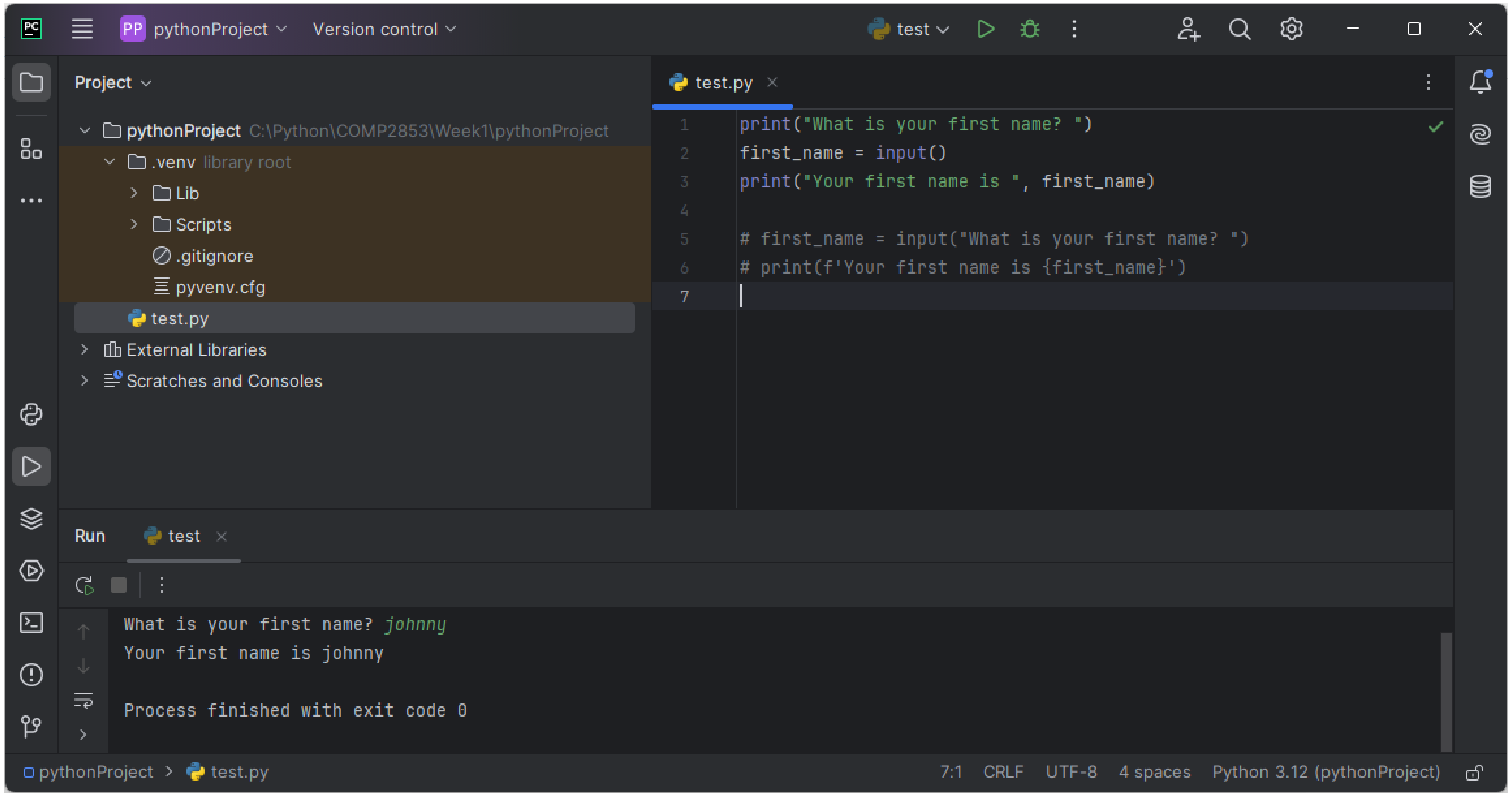
Our First Python Program

- Your project and files are stored in the filesystem by default (Do not store there)
- .idea – PyCharm IDE settings for your project
- venv – The Python environment for your project

Our First Python Program

```
# my first-ever python script
# these comments are ignored by the python interpreter

print("What is your first name?")
first_name = input()

print("Your first name is", first_name)
```

Our First Python Program



```python
print("What is your first name? ")
first_name = input()
print("Your first name is ", first_name)

# first_name = input("What is your first name? ")
# print(f'Your first name is {first_name}')
```

```
What is your first name? johnny
Your first name is johnny

Process finished with exit code 0
```

Our First Python Program - Explained

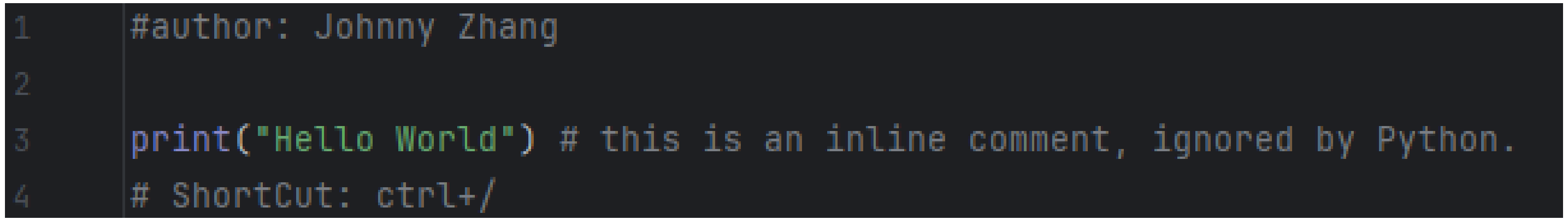
# my first-ever python script *# these comments are ignored by the # python interpreter*	Comments are for people to read. They are ignored by the Python interpreter. The hash tag # followed by a whitespace character tells the interpreter to ignore everything from here until the end of this line. You can also comment out part of a line.
print("What is your first name?")	Double-quotation and single-quotation marks are the same. print() is a built-in Python instruction. It writes text to the console (i.e., screen).
first_name = input()	input() is another built-in Python instruction. It reads input from the keyboard and returns the user’s input as a String. It could also have been written as: first_name = input(“What is your first name?”) first_name is a string variable; compound words are separated by an underscore.
print("Your first name is", first_name)	This prints the string “Your first name is” followed by space followed by the <u>value</u> of the first_name variable followed by a newline character.

Learning Outcomes: Lesson 1

- Variables
- Naming Conventions
- Built-in functions
- Expressions
- Types of Errors

Comments

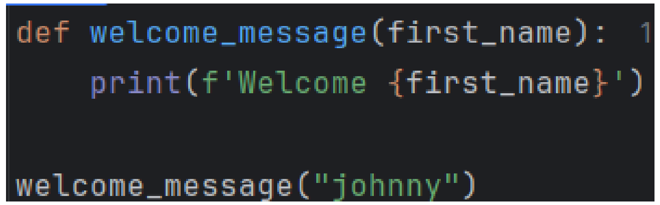
- Comments are used to explain and document your code, making it easier for *others (or yourself)* to read (*not* for the computer to read)
- They are not interpreted or executed by the python interpreter
- Single-line Comments: start with a # symbol and last until the end of that line
- Multi-Line Comments (Block Comments): In Python, there is no specific syntax for multi-line comments like in some other languages (e.g., /* */ in C/C++). However, multi-line comments can be written by using # at the beginning of each line. (PyCharm shortcut: ctrl+/)
- **In this course, always include a comment at the top of every file with your name**



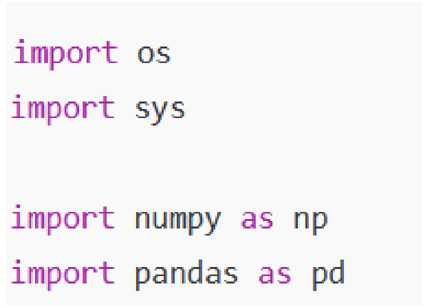
```python
#author: Johnny Zhang

print("Hello World") # this is an inline comment, ignored by Python.
# ShortCut: ctrl+/
```

Style

- Be readable and be consistent in how you code Python.
- https://www.python.org/dev/peps/pep-0008
- Consistency and readability and conventions are very, very important.
- Indentation:
    - Use 4 space per indentation level
    - Never mix spaces and tabs for indentation
    - Most IDEs and text editors automatically insert spaces when you press the Tab key.
- Line Length: Limit all lines to 79 characters or fewer (comments or docstrings).
- Blank lines: Use blank lines to separate functions, classes, and blocks of code inside function
- Imports: Imports should be on separate lines. Standard library imports should be grouped first, followed by third-party imports, then local imports.
- Naming conventions, Docstrings, Comments Etc.



```python
def welcome_message(first_name):
    print(f'Welcome {first_name}')

welcome_message("johnny")
```



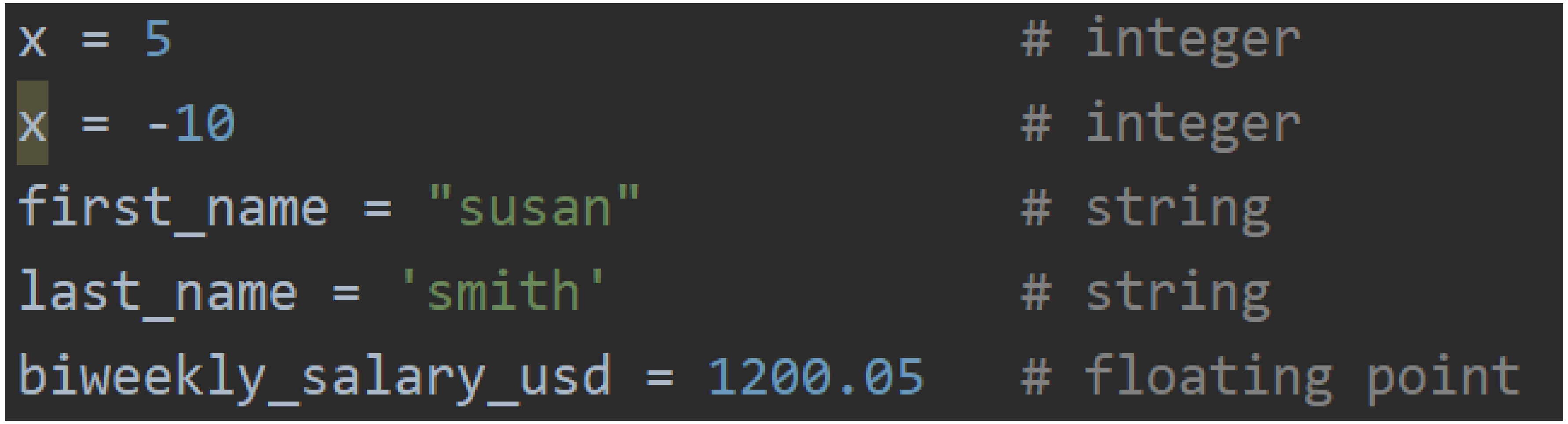
```python
import os
import sys

import numpy as np
import pandas as pd
```

Variables – Values and Types

- A variable is a name (label) that represents a value (with a data type) stored in the computer’s memory
- Values have different types such as:
  - **String** anything in "quotation marks" or 'quotation marks' such as "Hello World!" or 'this' or '5'
  - **Integer** whole numbers such as 10, 567964, 101, -66, 0
  - **Floating point** decimal numbers such as 11.65, -0.001, 1200.0, 0.0
  - **Boolean** True or False (vs. C, C++ JavaScript true & false)
- Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.
  - x = 5
- An assignment statement gives variables a value using a single = equals sign
  - school_name = "bcit" # use lowercase letters and the_underscore for compound names
  - print(**type**(school_name)) # str the **type()** instruction tells what type a variable is

Variables - Examples

- The data type of the variables is based on the value stored in the variable

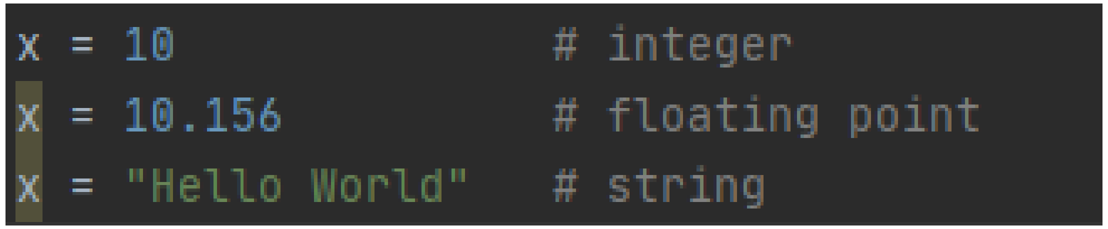


```
x = 5                           # integer
x = -10                         # integer
first_name = "susan"            # string
last_name = 'smith'             # string
biweekly_salary_usd = 1200.05   # floating point
```

- Note that in python, “double quotation marks” and ‘single quotation marks’ have the same meaning.

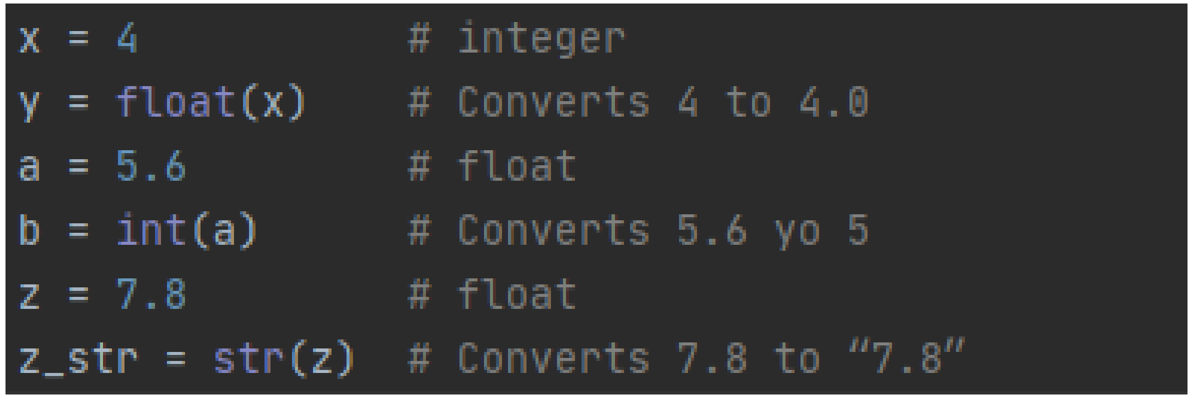
Variables - Types

- The type of the variable will change if the type of the value changes

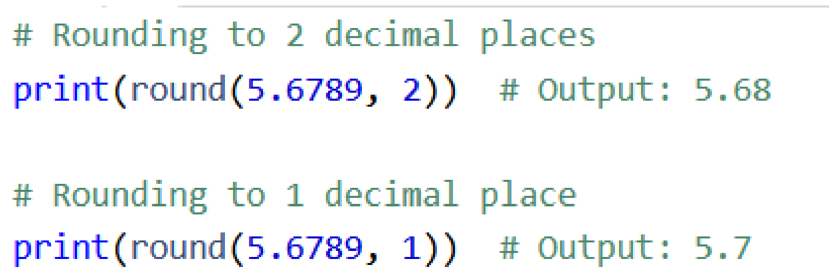


```
x = 10              # integer
x = 10.156          # floating point
x = "Hello World"   # string
```

- The type of a variable can be changed using built-in type functions (note: use this rarely)(note: round(number, ndigits) vs. round(number))



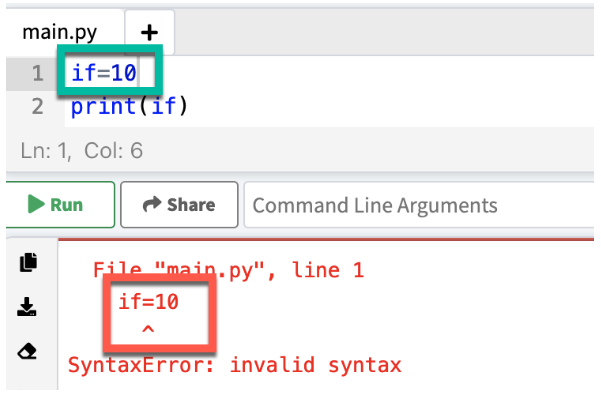
```
x = 4           # integer
y = float(x)    # Converts 4 to 4.0
a = 5.6         # float
b = int(a)      # Converts 5.6 yo 5
z = 7.8         # float
z_str = str(z)  # Converts 7.8 to “7.8”
```



```
# Rounding to 2 decimal places
print(round(5.6789, 2))  # Output: 5.68

# Rounding to 1 decimal place
print(round(5.6789, 1))  # Output: 5.7
```

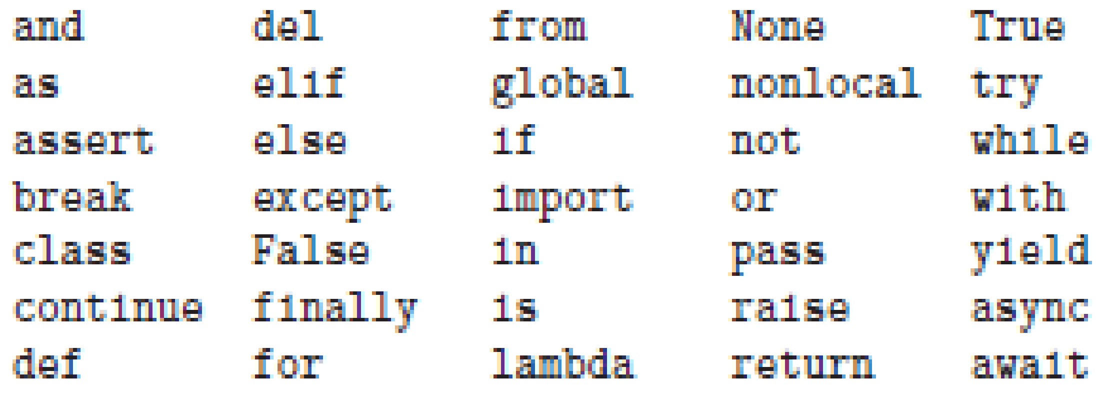
Naming Conventions

- Choose meaningful names that document what the variable is used for
- Can contain both letters, numbers and the underscore character but cannot start with a number
- Cannot use reserved Python keywords[AKA, keywords. Reserved words are special identifiers that Python uses to define the syntax and structure of the language.]; for example, these:



```
if=10
print(if)
```

```
File "main.py", line 1
  if=10
    ^
SyntaxError: invalid syntax
```



and	del	from	None	True
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	async
def	for	lambda	return	await

Naming Conventions

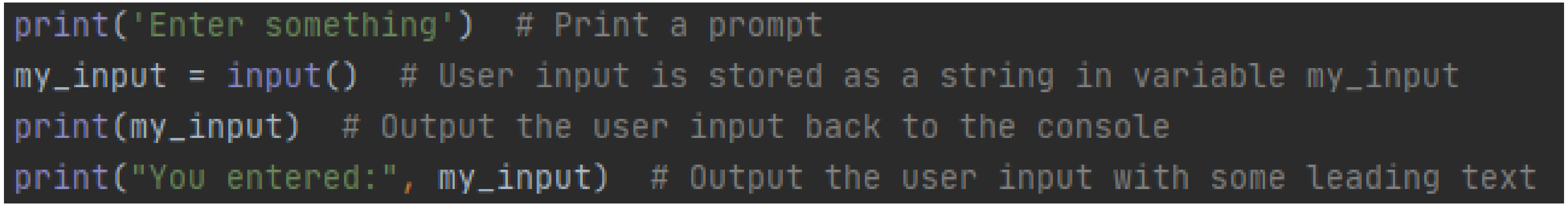
- Variable Naming
  - Variable names should be a descriptive noun or noun combination.
  - camelCase: first word starts with lower case and each consecutive word starts with upper case.
  - Variable names cannot contain spaces.
  - **Ex:** firstName = “Jane” or ‘Jane’

Naming Conventions

- Use **lower_snake_case (Snake Case)** for variables, modules and function names, like these:
  - **my_input, first_name, last_name, weight_kg, year_born, print_result**
- Use **UPPER_CASE (Uppercase Snake Case)** for constants (i.e. whose values will never change vs. JavaScript const)
  - **PI = 3.14, LEAGUE_NAME = “Fifa”**
- Use UpperCamelCase only for classes (covered much later)
  - **BankAccount, CoinFlip, Animal, UserProfile**

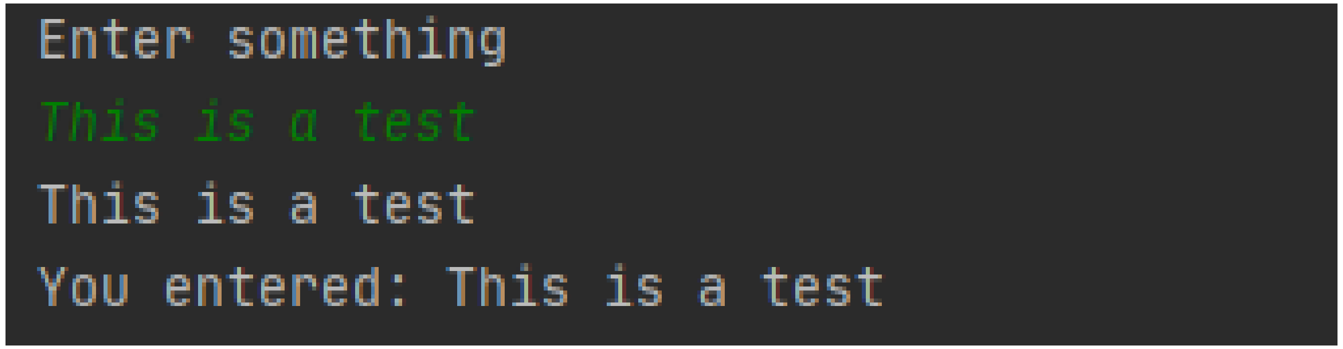
Built-in Functions

- Python can read input from the keyboard using the built-in **input()** instruction
- Python can print to the console using the built-in **print()** instruction



```
print('Enter something')  # Print a prompt
my_input = input()  # User input is stored as a string in variable my_input
print(my_input)  # Output the user input back to the console
print("You entered:", my_input)  # Output the user input with some leading text
```

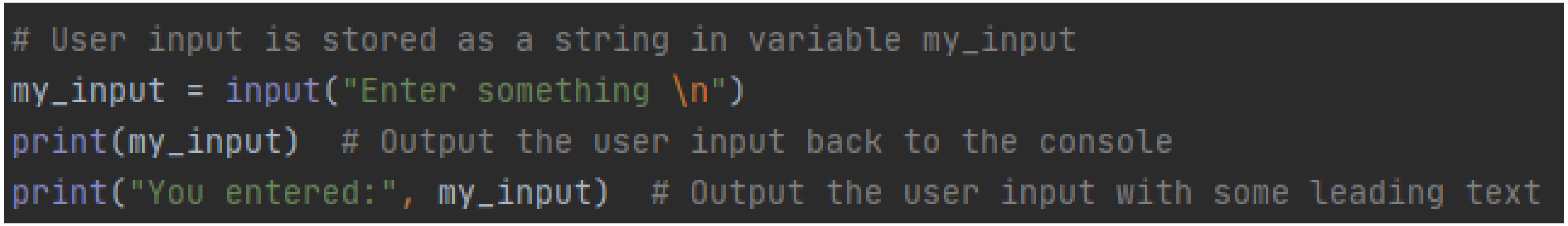
After running this program, the console would look like this:



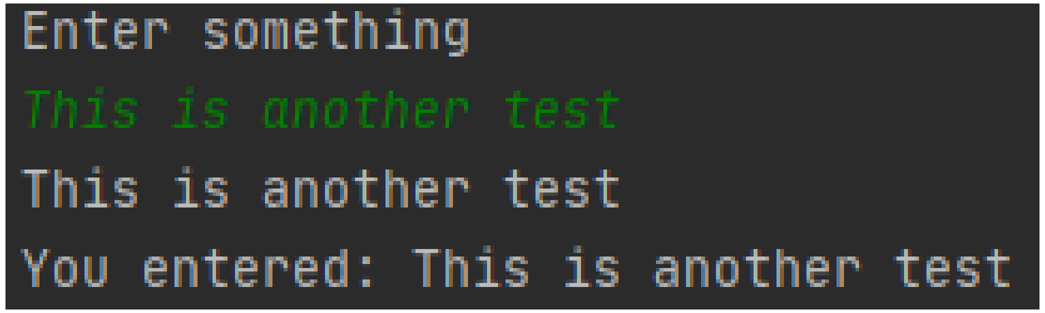
```
Enter something
This is a test
This is a test
You entered: This is a test
```

Built-in Functions

- Reduce code:
- The **input()** instruction can also print a prompt before reading keyboard input: (note: **n** means newline)
- Console



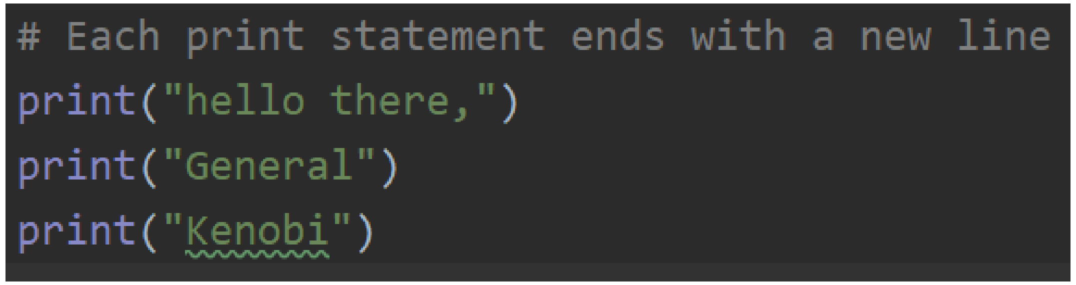
```
# User input is stored as a string in variable my_input
my_input = input("Enter something \n")
print(my_input)  # Output the user input back to the console
print("You entered:", my_input)  # Output the user input with some leading text
```



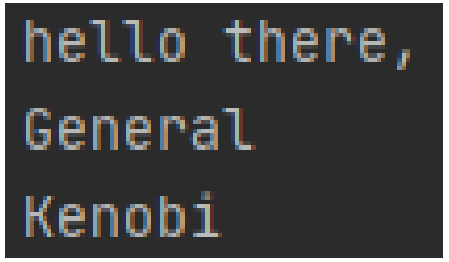
```
Enter something
This is another test
This is another test
You entered: This is another test
```

Built-in Functions

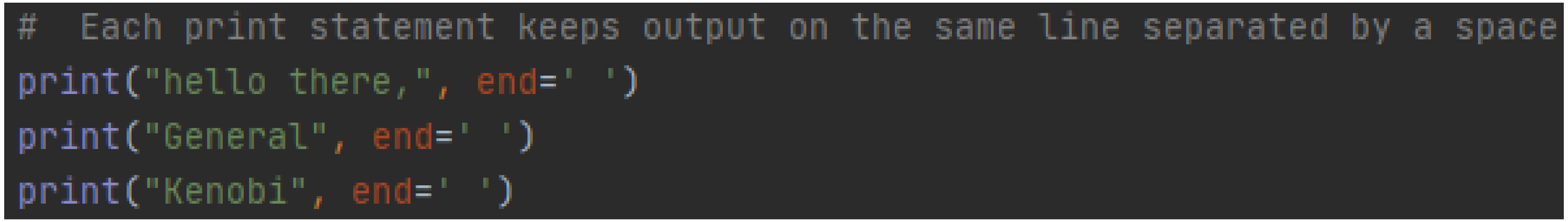
- Output to the console (i.e. the screen) using the built-in function print() terminates the print with a newline character (i.e. acts like the user pressed Enter on the keyboard).
- BUT: you can change it, so the end is not a newline, but is something different:



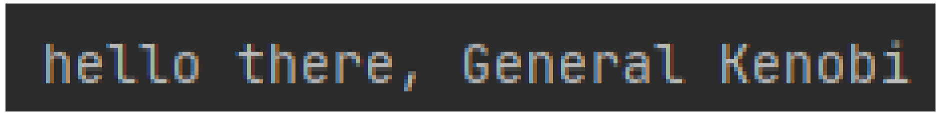
```python
# Each print statement ends with a new line
print("hello there,")
print("General")
print("Kenobi")
```



```
hello there,
General
Kenobi
```



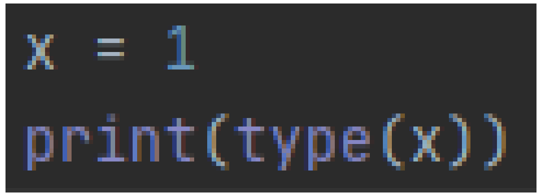
```python
#  Each print statement keeps output on the same line separated by a space
print("hello there,", end=' ')
print("General", end=' ')
print("Kenobi", end=' ')
```



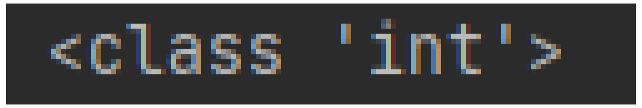
```
hello there, General Kenobi
```

Built-in Functions

- Everything in python is an object, including variables like integers and strings. More will be discussed later on objects.
- The built-in function **type()** will return the data type of an object

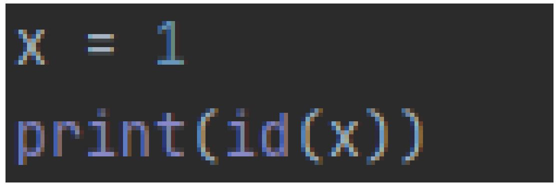


```
x = 1
print(type(x))
```

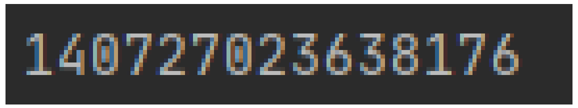


```
<class 'int'>
```

- The built-in function **id()** will return the id of an object, an integer that uniquely identifies the object in a program



```
x = 1
print(id(x))
```

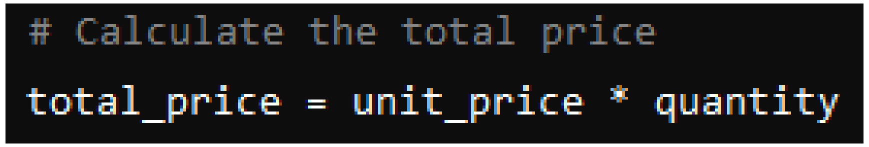


```
140727023638176
```

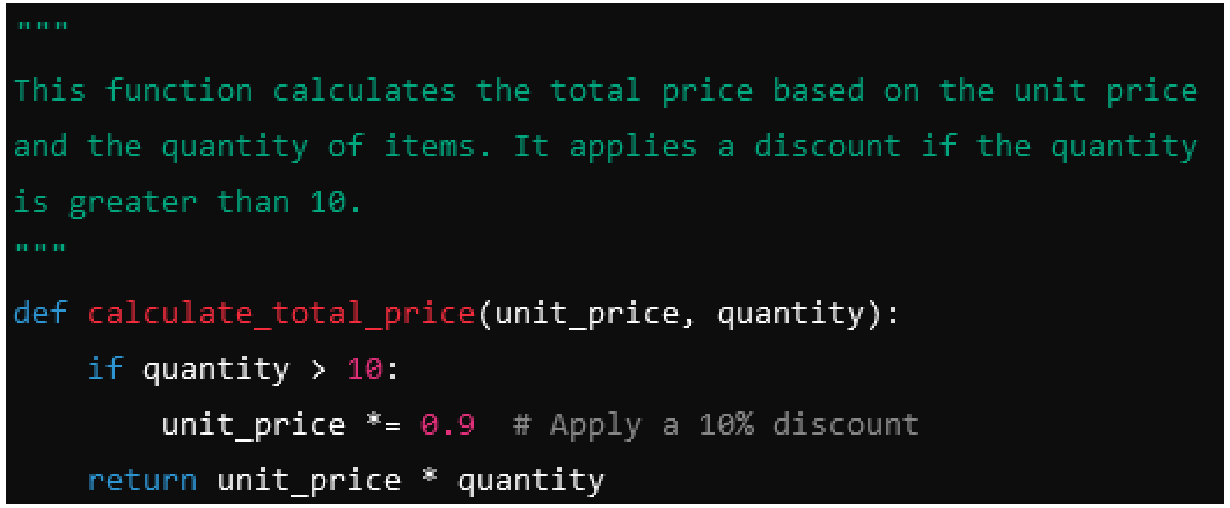
Note: JavaScript typeof

Comments

- Comments
  - Comments help to narrate your code.
  - The Python compiler ignores them.
  - Single-Line Comments begin with “#”
  - Multi-Line Comments use triple quotes (''' or """)



```
# Calculate the total price
total_price = unit_price * quantity
```



```
"""
This function calculates the total price based on the unit price
and the quantity of items. It applies a discount if the quantity
is greater than 10.
"""
def calculate_total_price(unit_price, quantity):
    if quantity > 10:
        unit_price *= 0.9  # Apply a 10% discount
    return unit_price * quantity
```

Comments: Best Practices

- Be Concise and Clear: Write comments that are easy to understand and directly related to the code they describe. Avoid unnecessary or redundant comments.
- Update Comments: Ensure comments are updated along with code changes to prevent outdated or misleading information.
- Use Comments to Explain Why, Not What: The code should be self-explanatory regarding what it does. Use comments to explain why certain decisions were made or why certain methods are used.
- Avoid Over-Commenting: Don’t comment on every single line of code. Focus on the sections where explanation is necessary.
- Avoid Commenting Out Code: Instead of commenting out code, consider using version control systems (like Git) to keep track of code changes.

Python Casting

- Casting in python is therefore done using constructor functions:
  - ✓ int() - constructs an integer number from an integer, a float (by removing all decimals), or a string (providing the string represents a whole number)
  - ✓ float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
  - ✓ str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

```
# Comments narrate your code. They are not compiled with the
# Python instructions. Comments begin with '#'.
# The variable 'age' stores a whole number.
age = 25
# The variable 'firstName' stores a String which is any character
combination
# within quotes.
firstName       = "Jane"
# The variable 'earnings' stores a floating-point number.
earnings = 15.22
# The variable 'isBCITStudent' storas a Boolean value. Booleans in
# Python can be either True or False. They are useful for implementing
# logic.
isBCITStudent   = True
print("Age: "        + str(age))
print("First Name: " + firstName)
print("Earnings: "   + str(earnings))
```

Arithmetic Operators

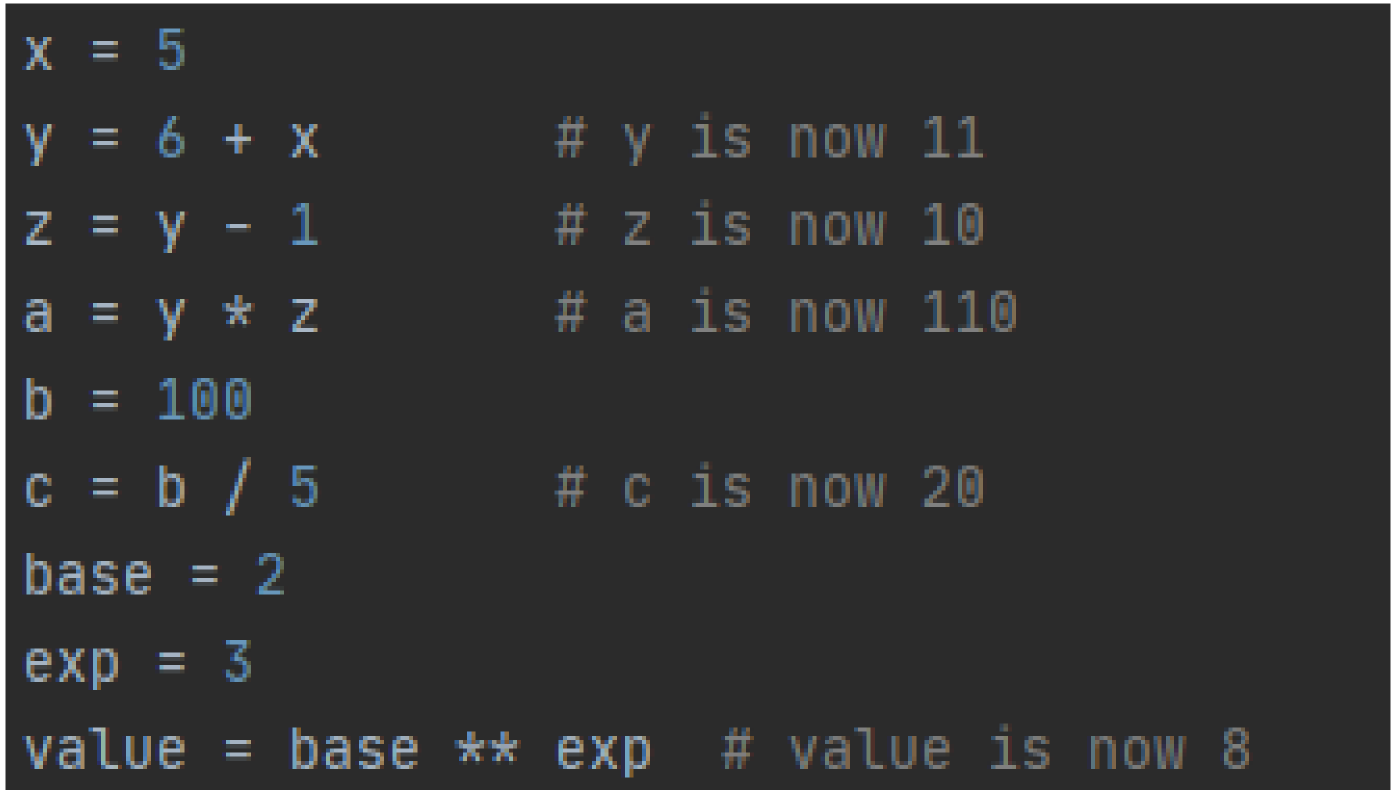
- Python provides a set of arithmetic operators to perform various mathematical operations.

Arithmetic Operators in Python

+	Addition (x + y)
-	Subtraction (x – y)
*	Multiplication (x * y)
/	Division (x / y)
//	Integer Division (result will be a whole number; any decimal portion is thrown away, not rounded)
**	Exponent (x to the power of y)
%	Modulus (remainder after dividing)

Arithmetic Operators

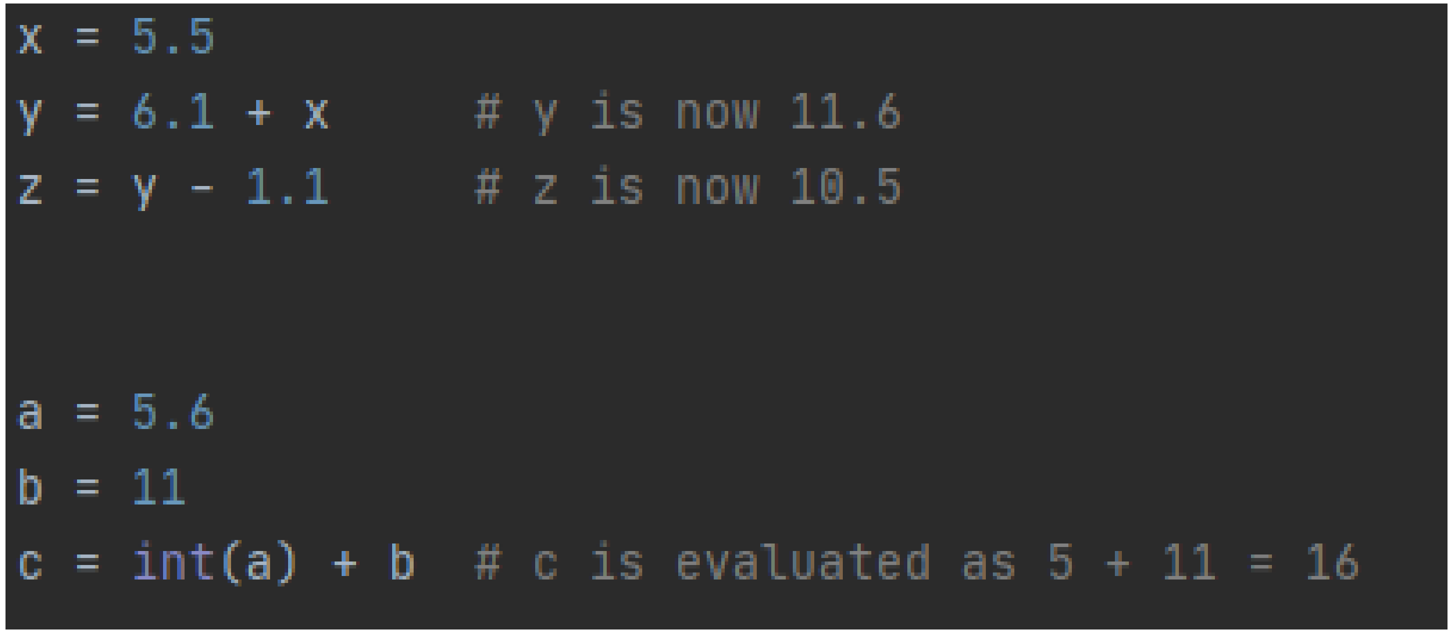
- Some examples



```
x = 5
y = 6 + x       # y is now 11
z = y - 1       # z is now 10
a = y * z       # a is now 110
b = 100
c = b / 5       # c is now 20
base = 2
exp = 3
value = base ** exp  # value is now 8
```

Arithmetic Operators

- Some more examples



```
x = 5.5
y = 6.1 + x     # y is now 11.6
z = y - 1.1     # z is now 10.5



a = 5.6
b = 11
c = int(a) + b  # c is evaluated as 5 + 11 = 16
```

- print(3*6**2/6) ?

Categories of Errors

- Syntax – The rules of the programming language
- Syntax Error – A violation of the programming language rules. Usually caught by the interpreter before any lines of the program are run.
- Run your program frequently to catch these errors, especially while you are new to programming
- Very common error for new programmers is a typo. Make sure everything is spelled correctly and names match (Python is case sensitive… **school_name** is a different variable than **School_Name**, etc…)

Categories of Errors

- Runtime – When the program is run by the interpreter
- Runtime Error – Correct syntax but the program attempts to do something that is impossible such as divide by zero or trying to multiply strings.
- The program immediate stops and reports and error at the line of code being run so it is often called a crash. Common Runtime Error Types:
    - SyntaxError – Code that cannot be understood (but was not caught at startup)
    - IndentationError – Lines of the program are not indented correctly
    - ValueError – Invalid value is used (i.e., passing a string value to int())
    - NameError – Program tries to use a variable that does not exist (i.e., print(x))
    - TypeError – An operation uses incorrect types (i.e., adding an integer to a string)

Categories of Errors

- Semantic Error – Your program runs without any syntax errors, but does not produce the expected or correct result due to incorrect logic or misunderstanding of how a particular part of the program should work. Also known as a Logic Error.

Examples:

- Your program is supposed to add x and y and print the result, but always prints the value of x
- Your program is supposed to load in data from a file, but the variable holding that data is always empty

Incorrect Logic

```
def calculate_area(radius):
    return 2 * 3.14 * radius  # Incorrect formula for area

print(calculate_area(5))  # Expected 78.5, but this calculates the circumference
```

Variable Misuse

```
def greet(name):
    greeting = "Hello, " + name
    return greeting

name = 5
print(greet(name))  # Error: concatenation of str and int
```

Function Logic Errors

```
def divide(a, b):
    return a / b

result = divide(10, 0)  # Semantic error: division by zero
```

Programming Best Practices/Advice

- Make your code readable to others:
  - Use comments to describe why you’ve done something
  - Use whitespace (e.g., blank lines) to group related bunches of code
  - Put units on variable names (e.g. **weight_kg** instead of **weight**)
- Don’t be afraid to ask for clarification… but do try to understand things yourself, first
- Don’t be afraid to experiment with your code
  - You can’t cause any serious harm (generally)
  - You can always recover
  - Worst case: you start writing your program over again (for simple programs)

Lab 1 and Lesson 2

We will take a break and then start on Lab 1

- Please complete Part A and send me your screenshot to verify you have your PyCharm IDE setup correctly
- Complete Part B and submit to the Lab 1 Dropbox (Assignments -> Lab 1). It will provide you feedback next class.

Lab 1 is due midnight before next class in the Lab 1 Dropbox on the Learning Hub

Everyone submits their own work. Once you are finished the lab, you are done for today’s class