**Introduction to Python WEEK 1**

**DAY 1**

Python's Growth: Python is experiencing significant growth in the job market. Knowledge of Python can open doors to top positions in just a few years.  
  
Continuous Learning: To excel in the Python job market, continuous learning is essential. Expanding your knowledge of Python and related technologies is crucial.  
  
Versatility: Python is versatile and can be applied in a wide range of fields, making it a valuable skill to have.  
  
Cross-Language Skills: While Python is important, having a working knowledge of other programming languages can be beneficial. Courses and training in other languages, like those provided by CapaCiTi, can enhance your qualifications.  
  
Python and MySQL Integration: Python is often used to integrate applications with MySQL. Many companies are transitioning from expensive systems to open-source solutions, which is creating a demand for Python skills in this context.  
  
Prominent Python Users: Companies like YouTube and BitTorrent use Python, demonstrating its effectiveness in real-world applications.  
  
Network Programming: Python is also used in network programming, but it requires an in-depth understanding of network control and management.  
  
Various Job Fields: Python offers opportunities in various job fields, including Software Engineering, Software Development, Research Analysis, Data Analysis, and Data Science. Database experience is often required in these roles.  
  
Short-Term Courses: Some courses aim to provide an intensive, short-term learning experience to equip individuals with the programming knowledge required for Python-related job positions.

#### Daily Notes - Activity 1 - Installing Anaconda on Windows

Download Anaconda:  
  
Go to the Anaconda website at <https://www.anaconda.com/products/distribution> and download the Anaconda Distribution for Windows. Make sure to download the version that corresponds to your Windows system (32-bit or 64-bit).  
  
Run the Installer:  
  
Once the installer is downloaded, locate the installer file (it should be named something like Anaconda3-<version>-Windows-x86[\_64].exe) and double-click it to run the installer.  
  
Accept License Agreement:  
  
Read and accept the license agreement.  
  
Select Installation Type:  
  
You'll be presented with two installation options:  
  
Install for just me (recommended): This will install Anaconda for your user profile.  
Install for all users: This will require administrative privileges and make Anaconda available to all users on the system.  
Choose the option that suits your needs and click "Next."  
  
Choose Installation Location:  
  
You can choose the installation location or stick with the default. If you want to change it, click "Browse" and select the desired folder. Click "Next" to continue.  
  
Advanced Options (Optional):  
  
You can optionally add Anaconda to your system's PATH environment variable. It's usually a good idea to check this option as it makes it easier to use Anaconda from the command line.  
  
Install:  
  
Click the "Install" button to begin the installation process.  
  
Installation Progress:  
  
The installer will start installing Anaconda on your system. This may take a few minutes.  
  
Installation Complete:  
  
Once the installation is complete, you'll see a "Completion" screen. Make sure the box that says "Add Anaconda to my PATH environment variable" is checked if you didn't do so in the advanced options. Click "Next."  
  
First Time Use:  
  
You can choose to install Visual Studio Code or not. This is optional and depends on your preference. You can always install VS Code later if you don't want it at this point.  
  
Finish:  
  
Click "Finish" to exit the installer.  
  
Testing Your Installation:  
  
To test your installation, open the Anaconda Navigator from the Start menu or open a command prompt or Anaconda Prompt and run the command conda list. This command should display a list of installed packages.

#### Daily Notes - History of Python

Python was conceived in the late 1980s and Guido van Rossum started implementing it at CWI in the Netherlands in December 1989. It is a relatively simple language that includes a standard library that provides modules for a large number of processes that programs deal with. This approach keeps Python simple yet reliable programming language.  
  
Python is implemented in C and relies on the extensive, well understood, portable C libraries. It fits seamlessly with UNIX, Linux, and POSIX environments. Since these standard C libraries are widely available for the various MS-Windows variants, and other non-POSIX operating systems, Python runs similarly in all environments. The Python programming language was created based on lessons learned during language and operating system support. Python is built from concepts in the ABC and Modula-3 languages.

#### Daily Notes - Invoking the Interpreter

Python was conceived in the late 1980s and Guido van Rossum started implementing it at CWI in the Netherlands in December 1989. It is a relatively simple language that includes a standard library that provides modules for a large number of processes that programs deal with. This approach keeps Python simple yet reliable programming language.  
  
Python is implemented in C and relies on the extensive, well understood, portable C libraries. It fits seamlessly with UNIX, Linux, and POSIX environments. Since these standard C libraries are widely available for the various MS-Windows variants, and other non-POSIX operating systems, Python runs similarly in all environments. The Python programming language was created based on lessons learned during language and operating system support. Python is built from concepts in the ABC and Modula-3 languages.

#### Daily Notes - Activity 2 - Interactive Mode

Interactive mode in Python is a way of interacting with the Python interpreter one command at a time. It allows you to enter and execute Python code statements or expressions interactively, making it useful for testing and experimenting with code. You can use Python's interactive mode by running the Python interpreter in your command prompt or terminal.  
  
  
Interactive mode in Python is a way of interacting with the Python interpreter one command at a time. It allows you to enter and execute Python code statements or expressions interactively, making it useful for testing and experimenting with code. You can use Python's interactive mode by running the Python interpreter in your command prompt or terminal. Here's how to access and use interactive mode:  
  
Open a Command Prompt or Terminal:  
  
On Windows, you can open the Command Prompt by searching for "cmd" in the Start menu. On macOS or Linux, you can use the Terminal.  
  
Start the Python Interpreter:  
  
To start the Python interpreter in interactive mode, simply type python (or python3 on some systems) and press Enter. You should see the Python prompt, which typically looks like >>>.  
  
C:\Users\YourUsername> python  
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 24 2020, 17:52:18)  
[GCC 8.4.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>>

#### **Comments in Python**

Comments in Python start with the hash character, #, and extend to the end of the physical line. A comment may appear at the start of a line or following whitespace or code, but not within a string literal. A hash character within a string literal is just a hash character. Since comments are to clarify code and are not interpreted by Python, they may be omitted when typing in examples.

#### Daily Notes - Activity 3 - Using Python

Readability and Simplicity: Python's clean and easy-to-read syntax makes it an excellent choice for beginners and experienced developers alike. Its use of indentation to define code blocks enforces a consistent and readable coding style.  
  
Extensive Standard Library: Python comes with a vast standard library that provides modules and packages for a wide range of tasks, reducing the need to write code from scratch and saving development time.  
  
Cross-Platform Compatibility: Python is available on various platforms, including Windows, macOS, and Linux, making it a highly portable language.  
  
Large and Active Community: Python has a large and active community of developers who contribute to its ecosystem, provide support, and create third-party packages and frameworks.  
  
Third-Party Libraries and Frameworks: Python boasts a rich ecosystem of third-party libraries and frameworks for a wide range of applications, including web development (Django, Flask), data analysis (NumPy, pandas), machine learning (TensorFlow, PyTorch), and more.  
  
Open Source: Python is open source, meaning it's free to use, distribute, and modify. This encourages innovation and the creation of a vast number of Python-based projects.  
  
Interoperability: Python can easily interface with other programming languages like C, C++, and Java, making it an excellent choice for extending existing codebases and working with external libraries.  
  
Rapid Development: Python's simplicity and extensive libraries enable developers to build applications and prototypes quickly, which is particularly beneficial for startups and projects with tight deadlines.  
  
Strong Support for Scientific Computing and Data Analysis: Python is widely used in scientific computing and data analysis due to libraries like NumPy, SciPy, and pandas, as well as visualization tools like Matplotlib and Seaborn.  
  
Robust Web Development: Python's web frameworks, such as Django and Flask, are popular for building web applications, and Python is also commonly used for server-side scripting.  
  
Automation and Scripting: Python is an excellent choice for writing scripts to automate repetitive tasks, making it valuable for system administration, data processing, and other automation needs.  
  
Community and Learning Resources: Python has a wealth of tutorials, documentation, and online courses, making it an accessible language for learners and a resource-rich environment for developers.  
  
Scalability: Python is suitable for small scripts and large, complex applications, making it a versatile choice for various project sizes.  
  
Versatility: Python can be used for a wide range of applications, from web development and data analysis to scientific computing, machine learning, artificial intelligence, and more.

#### **DAY 2 Introduction to Variables**

Variables are a temporary storage space in a computer’s memory and their re needed to assign values that we use to execute on the code. Also for programs that are case sensitive we need to check that and be more careful into naming the variables.

Variables are a temporary storage space in a computer’s memory.  
When a variable’s value changes the program’s current state also changes.

A variable acts as a container to hold a different number of data items or values.  
All programming languages use variables, as they are among the most important elements in programming, and that is why a good understanding of variables will only make your job easier when writing programs.

Variables are also used to move data between functions; this will be discussed later.  
  
Every variable is created with an initial value. A variable can be in three states:  
  
Variable creation (Declaration)  
Variable assignment (Initialization)  
Variable changed (Execution)  
  
In Python identifiers are case sensitive, so for example, firstName, FirstName, FIRSTNAME, and firstname are four different identifiers.  
A second rule is that variables cannot have the same name as Python’s keywords.  
We can find out what keywords are in Python, by using the function called dir(). If this function is called with the \_\_builtins\_\_ attribute, it returns a list of Python’s built-in attributes.

#### **Using variables**

A variable is assigned automatically to an appropriate data type. For example, Python automatically assigns a variable to a string data-type, if an input or value is given that contains letters or words.  
Values of the same type can be manipulated together. Sometimes Python finds a way to manipulate values into a common type by casting the values automatically.  
There are cases where values need to be cast explicitly. The example code below illustrates how values are assigned to variables.

**Casting**

The rules to convert a string to a float are:  
  
The string should only contain numbers.  
Other than numbers the following are allowed:  
Only one dot (.) character. Indicates the decimal starts after the dot (.) character.  
A ‘+’ or ‘−‘ character at the beginning of the string. This indicates that the number is either positive or negative.  
  
  
**The rules to convert a string to a float are:**  
  
The string should only contain numbers.  
Other than numbers the following are allowed:  
Only one dot (.) character. Indicates the decimal starts after the dot (.) character.  
A ‘+’ or ‘−‘ character at the beginning of the string. This indicates that the number is either positive or negative.

**DAY 3 DATA TYPE**

Data types in programming are used to define the kind of data that can be stored and manipulated in a programming language.  
  
Different programming languages may have additional data types or variations of these common data types. Understanding data types is crucial for writing correct and efficient code, as it helps ensure that operations and manipulations are done in a way that aligns with the intended use of the data. Data types also dictate the kind of operations that can be performed on the data, such as arithmetic, string manipulation, and comparisons.

**Integers**

These represent numbers in an unlimited range. This is only limited by a machine’s memory.  
  
Booleans: Evaluate to ‘True or False’, 1 or 0 respectively.  
  
Floating point numbers: Floating-point numbers represent double-precision numbers.  
  
Complex numbers: Complex numbers represent numbers as a pair of double-precision numbers.  
  
Strings: A sequence of Unicode characters e.g. a word or a sentence that can be manipulated.

Integers, often represented as `int` in programming, are a fundamental data type used to represent whole numbers without a fractional or decimal component. Here are some key characteristics and uses of integers:  
  
1. \*\*Whole Numbers:\*\* Integers can represent both positive and negative whole numbers, including zero.  
  
Example:  
- Positive integers: 1, 2, 3, 100  
- Negative integers: -1, -2, -3, -100  
- Zero: 0  
  
2. \*\*Mathematical Operations:\*\* Integers can be used in various mathematical operations, such as addition, subtraction, multiplication, and division.  
  
Example:  
- Addition: 5 + 3 = 8  
- Subtraction: 10 - 7 = 3  
- Multiplication: 4 \* 6 = 24  
- Division: 8 / 2 = 4  
  
3. \*\*Integer Division:\*\* When two integers are divided, the result is also an integer. Any fractional part is truncated (not rounded).  
  
Example:  
- Integer division: 7 // 2 = 3 (not 3.5)  
- Remainder: 7 % 2 = 1  
  
4. \*\*Common Uses:\*\* Integers are used in a wide range of applications, including counting, indexing, iteration, and representing quantities, such as the number of items in a list or the age of a person.  
  
5. \*\*Size:\*\* The size of an integer can vary depending on the programming language and system architecture. In many programming languages, including Python, integers have a fixed size, which means there's a maximum and minimum value they can represent.  
  
6. \*\*Type Conversion:\*\* You can convert other data types, such as floating-point numbers or strings, to integers using casting or type conversion functions like `int()`.  
  
Example:  
- Casting a float to an int: `int(3.14)` results in `3`.  
- Parsing a string to an int: `int("42")` results in `42`.  
  
7. \*\*Arithmetic Operations:\*\* Integers can be involved in various arithmetic operations, including addition, subtraction, multiplication, and division, which follow the rules of arithmetic.  
  
Example:  
- `3 + 5` results in `8`  
- `10 - 4` results in `6`  
- `2 \* 6` results in `12`  
- `20 / 5` results in `4`  
  
8. \*\*Comparison Operations:\*\* Integers can be compared using comparison operators like `==` (equal), `!=` (not equal), `<` (less than), `>` (greater than), `<=` (less than or equal to), and `>=` (greater than or equal to).  
  
Example:  
- `5 < 10` is `True`  
- `7 == 7` is `True`  
- `3 != 3` is `False`  
  
Integers are essential in programming for various tasks, including counting, indexing, and performing mathematical operations. They play a vital role in computer science and software development, making them a fundamental data type in most programming languages.

#### **Floating point numbers**

Floating-point numbers, often represented as float in programming, are a data type used to represent real numbers that include a fractional part. Here are some key characteristics and uses of floating-point numbers:  
  
Fractional and Decimal Numbers: Floating-point numbers can represent both whole numbers and numbers with fractional or decimal parts.  
  
Example:  
  
Whole numbers: 5.0, -7.0  
Decimal numbers: 3.14, -0.5  
Scientific notation: 1.2e6 (1.2 x 10^6)  
Mathematical Precision: Floating-point numbers provide a high degree of precision, allowing for accurate representation of real-world quantities, scientific values, and mathematical calculations.  
  
Range of Values: Floating-point numbers can represent a wide range of values, from very small (close to zero) to very large (e.g., astronomical or scientific values).  
  
Arithmetic Operations: Floating-point numbers can be used in various mathematical operations, such as addition, subtraction, multiplication, and division. These operations follow standard arithmetic rules.  
  
Example:  
  
Addition: 3.14 + 1.5 = 4.64  
Subtraction: 10.5 - 2.3 = 8.2  
Multiplication: 2.5 \* 3.0 = 7.5  
Division: 7.0 / 2.0 = 3.5  
Precision Limitations: Due to the limitations of computer hardware, floating-point numbers may not always represent real numbers with complete accuracy. There can be rounding errors or small discrepancies when performing operations.  
  
Special Values: Floating-point numbers include special values, such as positive and negative infinity (inf) and "not-a-number" (NaN), which are used to represent overflow, underflow, and undefined or indeterminate results.  
  
Example:  
  
Positive infinity: float('inf')  
Negative infinity: float('-inf')  
NaN: float('nan')  
Type Conversion: You can convert other data types, such as integers or strings, to floating-point numbers using casting or type conversion functions like float().  
  
Example:  
  
Casting an integer to a float: float(42) results in 42.0.  
Parsing a string to a float: float("3.14") results in 3.14.  
Comparison Operations: Floating-point numbers can be compared using comparison operators like == (equal), != (not equal), < (less than), > (greater than), <= (less than or equal to), and >= (greater than or equal to). However, due to precision limitations, comparisons may require tolerance thresholds.  
  
Example:  
  
3.14 < 3.1416 is True  
0.1 + 0.2 == 0.3 may not be exactly True due to rounding errors.  
Floating-point numbers are crucial in various scientific, engineering, and financial applications. They allow for the accurate representation of measurements, calculations, and real-world values that are not integers. However, when working with floating-point numbers, it's essential to be aware of potential precision issues and use appropriate techniques to mitigate errors, such as specifying a tolerance for comparisons.

#### Unpacking Argument Lists

String Manipulation: You can manipulate strings in various ways, such as converting them to uppercase or lowercase, replacing substrings, or splitting them into a list of words. The exact methods and functions available depend on the programming language.  
  
Escape Sequences: Strings can contain special characters that are represented using escape sequences. For example, "\n" represents a newline character, and "\t" represents a tab character.  
  
String Interpolation: Many programming languages allow you to insert variables or expressions into a string. This is often done using placeholders and string formatting functions.  
  
String Comparison: You can compare strings for equality or perform string comparisons to determine their order based on alphabetical or lexicographical order.  
  
String Methods: Most programming languages provide a range of built-in string methods for common operations like finding substrings, replacing text, or checking if a string starts or ends with a particular sequence.  
  
Immutable: In many programming languages, strings are immutable, meaning they cannot be modified once they are created. Any operation that appears to modify a string actually creates a new string.

#### Lambda Expressions

Lambda expressions, often simply referred to as "lambdas," are a feature in many programming languages that allows you to define anonymous, small, and inline functions.  
Lambda expressions are particularly common in functional programming languages and languages that support functional programming concepts.  
They are used to create functions on the fly for various purposes, such as passing functions as arguments to higher-order functions or for concise, one-time use cases.  
  
  
**lambda arguments: expression**  
  
add = lambda x, y: x + y  
result = add(3, 4) # result will be 7  
  
  
Lambda expressions are a versatile feature that enables you to write more readable and concise code, especially in scenarios where you need to pass functions as arguments or define simple, one-time-use functions. They are a key component of functional programming and are widely adopted in modern programming languages.

#### **Conventions about the content and formatting of documentation strings**

The first line should always be a short, concise summary of the object’s purpose. For brevity, it should not explicitly state the object’s name or type, since these are available by other means (except if the name happens to be a verb describing a function’s operation). This line should begin with a capital letter and end with a period.

**DAY 4**

#### **Introduction to operators**

Operators are used to testing conditions and manipulating values. Most statements contain expressions: an example of an expression is: 2 + 3.

When two objects of a different type, like str and int, are compared they are never equal, except for different numeric types like int and float which can be equal.

The <, <=, > and >= operators raise a TypeError exception when any operand is a complex number if the objects are different types that cannot be compared to one another, or where there is no defined ordering.

#### **Using Operators**

Operators provide the functionality to an expression and can be represented by symbols such as + or by keywords such as and. Operators require data to operate, and this data is known as an operand. In the example: 2 and 3 functions as operands for the + operator, and a and b function as the operands for the / (divide) operator.

#### **Activity 1**

import math  
numbers = 9.848787387  
roundNumbers=round(numbers)  
print(roundNumbers)  
  
print("Enter number of litres:" )  
numberOfLitres = float(input())  
numberOfBottles = numberOfLitres / 0.5  
remainingLitres = numberOfBottles % 0.5  
  
print( numberOfLitres,"L water will fill", math.trunc(numberOfBottles),"bottles",remainingLitres,"Litres remains")

**Activity 2**

#### Activity 2

# Constants  
MEGALITRE\_TO\_LITRE = 1\_000\_000  
DAM\_CAPACITY\_MEGALITRES = 1  
  
# Get the amount of water released from the user in millilitres  
water\_released\_millilitres = int(input("Enter the millilitres of water released to the oceans/rivers: "))  
  
# Convert millilitres to megalitres and calculate remaining water  
water\_released\_megalitres = water\_released\_millilitres / MEGALITRE\_TO\_LITRE  
water\_left\_megalitres = DAM\_CAPACITY\_MEGALITRES - water\_released\_megalitres  
  
# Convert remaining water back to litres  
water\_left\_litres = water\_left\_megalitres \* MEGALITRE\_TO\_LITRE  
  
# Calculate the percentage of water left  
percentage\_left = (water\_left\_litres / (DAM\_CAPACITY\_MEGALITRES \* MEGALITRE\_TO\_LITRE)) \* 100  
  
# Generate and display the report  
print(f"Report on Albasini Dam Water Conservation:")  
print(f"----------------------------------------")  
print(f"Water Released: {water\_released\_litres:.2f} litres")  
print(f"Water Remaining: {water\_left\_litres:.2f} litres")  
print(f"Percentage of Water Left: {percentage\_left:.2f}%")