**[Intro-to-Python-week-1-day-1](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1" \l "intro-to-python-week-1-day-1)**

[**Python in the Job Market**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#python-in-the-job-market)

Python is constantly growing in the market. With Python knowledge, you may find yourself in top positions within a few years, but you should increase your knowledge as much as possible. You will find that you often need a working knowledge of other programming languages, such as the language courses provided by CapaCiTi, to be qualified for a position in a company. This course covers important topics that you need to grasp the foundation of Python and make you ready for a position in the job market.

There are various job fields that Python has to offer. Integrating applications with MySQL is in high demand because MySQL and Python are both open-source applications. This means that some companies are switching over from their current expensive systems to open-source systems. YouTube and BitTorrent are examples of companies that use Python.

Network programming in Python is another option, which requires an extensive knowledge of how networking is controlled. Another fields for a Python programmer are Software Engineer, Software developer, Research Analyst, Data Analyst and Data Scientist; you will often be required to have database experience when working in this field.

This course is aimed at teaching you as much as possible in a relatively short time; its aim is to extend your programming knowledge further and introduce you to a new programming syntax.

[**History of Python**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#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 has an easy-to-use syntax that is focused on the programmer who must type in the program, read what was typed, and provide formal documentation for the program. Many languages have syntax focused on developing a simple, fast compiler; but those languages may sacrifice readability and can be more difficult to write. Python strikes a good balance between fast compilations and readability, and it is easier to write applications.

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.

[**Comments in Python**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#comments-in-python)

In programming, comments are a programming language construct used to insert human-readable text in the source code of a program. These extra pieces of text are ignored by the compiler and interpreter but can be potentially significant to programmers. Comments are added to make the source code easier to understand.

[**Comments could be used for a wide range of purposes, for example:**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#comments-could-be-used-for-a-wide-range-of-purposes-for-example)

Augmenting program code with basic descriptions to generate external documentation. Integration with source code management systems and other kinds of external programming tools. 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. Comments in Python are used to annotate your code with explanations or notes for yourself or other developers who might read your code.

[**Python has two types of comments:**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#python-has-two-types-of-comments)

[**Single-line Comments:**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#single-line-comments)

Single-line comments are used to add comments to a single line of code. They are created using the # symbol and everything after # on that line is considered a comment and is ignored by the Python interpreter. #This is a single-line comment x = 42 # This comment explains the purpose of the variable

[**Multi-line Comments (Docstrings):**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#multi-line-comments-docstrings)

Multi-line comments are often used for longer explanations, usually associated with functions, classes, or modules. These are created using triple double-quotes """ or triple single-quotes '''. Note that while multi-line comments are often used as docstrings for documentation purposes, they are also used for multi-line comments within functions or classes when necessary.

Comments are important for code readability and understanding, and they help other developers (or your future self) comprehend the purpose and functionality of your code. It's good practice to use comments to explain complex or non-obvious parts of your code.

[**Installing Anaconda on Windows:**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#installing-anaconda-on-windows)

* Visit Anaconda's website and download the Windows version.
* Choose the latest version for Python 3.x, usually 64-bit.
* Run the downloaded installer, accept terms, and avoid adding Anaconda to PATH.
* Access the Anaconda Prompt from the Start Menu to run Python code.

[**History of Python:**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#history-of-python-1)

* Created by Guido van Rossum in the late 1980s.
* Known for its simplicity and easy-to-read syntax.
* Offers a comprehensive standard library.
* Emphasizes clean, maintainable, and well-documented code.

[**Invoking the Interpreter:**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#invoking-the-interpreter)

* Python was created in the late 1980s for its simplicity and clear syntax.
* The language prioritizes the programmer's experience, readability, and maintainability.

[**Using Jupyter Notebook with Anaconda:**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#using-jupyter-notebook-with-anaconda)

* Launch Anaconda Navigator.
* Select Jupyter Notebook.
* Create a new Python notebook and start coding.

[**Comments in Python:**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#comments-in-python-1)

* Comments are denoted by the hash symbol (#) in Python.
* Used for documentation, clarification, and code readability.
* Improve code understanding and maintainability.

[**Creating Patterns in Python:**](https://github.com/andisiwenonkwenkwe/Intro-to-Python-week-1-day-1#creating-patterns-in-python)

* You provided code to create a pattern of characters, including asterisks and spaces.
* The code uses print statements and newline characters (\n) for spacing.
* The output generates a visually interesting pattern with a header and footer.

[**Variables in Python Day 2 week 1**](https://github.com/andisiwenonkwenkwe/Variables#variables-in-python-day-2-week-1)

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:**](https://github.com/andisiwenonkwenkwe/Variables#every-variable-is-created-with-an-initial-value-a-variable-can-be-in-three-states)

Variable creation (Declaration) Variable assignment (Initialization) Variable changed (Execution) Once the code which created the variable has finished executing, the variable is destroyed.

In Python, variables are defined in a standard way, by using the assignment character (=). This changes the value of the variable. Naming conventions specify the way in which variables should be named. This standard is used to make code more readable, and thus easier to understand.

The rules include the start and continuation characters. Variable names may contain any upper or lower case letter (A–Z, a–z), a number, or the underscore character. They may not begin with a number or contain spaces. Continuation characters are any characters except whitespace characters like tab and space.

[**Here are a few examples of valid variable names:**](https://github.com/andisiwenonkwenkwe/Variables#here-are-a-few-examples-of-valid-variable-names)

-c -ref\_number -admin -aVeryLongName

[**Here are a few examples of invalid variable names:**](https://github.com/andisiwenonkwenkwe/Variables#here-are-a-few-examples-of-invalid-variable-names)

True $name 12Graph 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. In programming, variables are used to store and manage data. They act as containers that hold values, allowing you to work with and manipulate those values within your program. Variables in Python are dynamically typed, meaning you don't need to specify their data type explicitly; Python infers it based on the value assigned to the variable. Here are some basics about variables in Python:

[**Variable Naming Rules:**](https://github.com/andisiwenonkwenkwe/Variables#variable-naming-rules)

Variable names can consist of letters, numbers, and underscores (\_). They must start with a letter or an underscore. Variable names are case-sensitive, so my\_variable and My\_Variable are considered different.

The **builtins** module contains all Python’s built-in attributes, which can be used with the dir()function. The ones that are returned are identified with the following characteristics:

Python’s built-in exceptions start with a capital letter. The rest are either functions or data type names. Identifiers that start and end with one or two underscores are special methods. All variables have to be assigned to a data type like a string (a series of characters) or an integer (positive and negative whole numbers). There are others, some of which will be discussed at a later stage.

Python has a powerful feature regarding the assignment. 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 automatically:

[**Casting**](https://github.com/andisiwenonkwenkwe/Variables#casting)

Casting can be done in two ways:

[**Implicitly:**](https://github.com/andisiwenonkwenkwe/Variables#implicitly)

The compiler automatically casts a value from one data type to another when assured that there will be no data loss. For example. casting from an integer variable to a floating-point variable or casting from an integer variable to another integer variable

[**Explicitly:**](https://github.com/andisiwenonkwenkwe/Variables#explicitly)

A value cannot be automatically cast from one data type to another if it will result in data loss. Extra code has to be written to ensure that the value stays the same and only the data type changes. For example, casting from a floating-point value to an integer value Notice that no errors occur when converting from an integer to a float. This will always be the case because an integer can be cast to a float data type implicitly because there will be no data loss. However the reverse is not true; a floating-point number cannot be implicitly cast into an integer as this will result in data loss (all data after the precision (.) will be lost), because the floating-point value does not get rounded off to the nearest whole number when implicitly converting to an integer (as seen in the above example).

[**The rules to convert a string to a float are:**](https://github.com/andisiwenonkwenkwe/Variables#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. In Python, casting, also known as type casting or type conversion, refers to the process of converting a value from one data type to another. This can be useful when you want to perform operations that involve variables of different data types or when you want to ensure that a variable has a specific data type. Python provides several built-in functions for type casting.

[**The most commonly used ones include:**](https://github.com/andisiwenonkwenkwe/Variables#the-most-commonly-used-ones-include)

int(): This function is used to convert a value to an integer data type. float(): This function is used to convert a value to a floating-point (decimal) data type. str(): This function is used to convert a value to a string data type. list(), tuple(), and dict(): These functions are used to convert values to list, tuple, and dictionary data types, respectively. bool(): This function is used to convert a value to a Boolean data type. In Python, most values are considered "truthy" except for False, None, 0, and empty collections (e.g., [], {}). Custom Type Casting: You can also create custom functions or methods to perform more complex type casting, especially when working with user-defined classes. Casting is essential for ensuring that your program handles data of different types correctly. It's important to be aware of data types and use type casting when necessary to avoid errors and ensure your program behaves as expected when working with various data sources and operations.

[**Variables as Temporary Storage:**](https://github.com/andisiwenonkwenkwe/Variables#variables-as-temporary-storage)

Variables store data temporarily in a computer's memory, holding various data types. 2. **States of Variables**: Variables go through creation, assignment, and potential change during program execution. 3. **Variable Destruction**: After the variable's scope finishes, it's destroyed, releasing memory. 4. **Python Variable Definition**: In Python, variables are created using the assignment operator (=). 5. **Naming Conventions**: Python has naming conventions for variables that enhance code readability. 6. **Case Sensitivity**: Python identifiers are case-sensitive. 7. **Reserved Keywords**: Python has reserved keywords that can't be used as variable names. 8. **Special Methods**: Identifiers starting and ending with underscores have special meanings in Python.

[**Casting in Python:**](https://github.com/andisiwenonkwenkwe/Variables#casting-in-python)

1. **Implicit Casting**: Python performs automatic casting without data loss.
2. **Explicit Casting**: You can explicitly cast values between data types.
3. **Concatenation vs. Addition**: The + operator behaves differently based on data types.
4. **Conversions between Data Types**: Converting between data types must consider potential data loss.
5. **Casting Strings to Floats**: Casting a string to a float has specific rules.

[**Using Variables:**](https://github.com/andisiwenonkwenkwe/Variables#using-variables)

* Variables are used to store and manipulate data in Python. They can hold different data types.

**Activity 1 - Consolidating Learnings:**

* The script calculates the sum of two numbers and then subtracts 3 from the total.
* It uses variables num1 and num2 for the numbers and displays the results with proper formatting.
* The script demonstrates variable assignment, arithmetic operations, and string concatenation.

**Activity 2 - Adding Totals as Characters:**

* This script is an extension of the previous one.
* It calculates and displays the sum and subtraction results as characters (strings) instead of numbers.
* It uses string concatenation to create the concatenated result.

[**Data-Types-in-Python-Day-3-week-1**](https://github.com/andisiwenonkwenkwe/Data-Types-in-Python-Day-3-week-1#data-types-in-python-day-3-week-1)

In programming, data types are a fundamental concept that defines the kind of data that a variable can hold or represent. They specify how data is stored in memory and what operations can be performed on that data. Data types help ensure data integrity and enable the compiler or interpreter to allocate memory and perform operations efficiently.

[**Common data types in programming include:**](https://github.com/andisiwenonkwenkwe/Data-Types-in-Python-Day-3-week-1#common-data-types-in-programming-include)

**Integer (int):** Represents whole numbers, positive or negative, without a decimal point. For example, 3, -42, 0.

**Floating-Point (float):** Represents real numbers with a decimal point. It includes numbers like 3.14, -0.5, and 2.0.

**String (str):** Represents a sequence of characters, such as text. For example, "Hello, World!" or "12345."

**Boolean (bool):** Represents two values, typically "True" or "False." Booleans are used for logical operations and comparisons.

**List:** Represents an ordered collection of values that can be of different data types. Lists are mutable, which means you can change their contents.

**Tuple:** Similar to a list but immutable. Once you create a tuple, you cannot change its elements.

**Dictionary:** Represents a collection of key-value pairs, where each key maps to a specific value. Dictionaries are unordered and mutable.

**Set:** Represents an unordered collection of unique elements. Sets are useful for performing operations like union, intersection, and difference.

**None:** Represents the absence of a value. It is often used to indicate that a variable has no value or that a function doesn't return anything.

**Custom Data Types (Classes and Objects):** In object-oriented programming, you can create your own data types by defining classes and objects.

The choice of data type is crucial in programming because it affects how memory is allocated, how operations are performed, and how data is handled. Using the right data type for a particular task is essential for writing efficient and error-free code.

Different programming languages may have additional data types and variations, and some languages allow you to define custom data types to suit your specific needs. The understanding of data types is fundamental for every programmer, as it dictates how data is processed and manipulated in a program. **1. Variables in Programming:**

* Variables are used to temporarily store data in a computer's memory.
* They go through three states: creation, assignment, and change.
* In Python, variables are defined using the assignment operator =.
* Naming conventions and case sensitivity in Python are important.
* Reserved keywords and special methods have naming restrictions.

**2. Data Types in Python:**

* Python supports various data types: integers, booleans, floating-point numbers, complex numbers, and strings.
* Choosing the right data type is crucial for program performance.
* Python automatically handles type conversions, making it dynamically typed.

**3. Working with Floating-Point Numbers:**

* Demonstrated formatting floating-point numbers in Python using format strings.
* Used %f to display full precision and %.<n>f to round to n decimal places.
* %+.2f was used to format with a plus sign for positive numbers.
* Conversion to octal, hexadecimal, and scientific notation was shown.

**4. Working with Strings:**

* Showcased creating and formatting strings in Python.
* Demonstrated string concatenation using + and using += for appending.
* Illustrated the use of escape sequences for line continuation.

**5. Lambda Functions in Python:**

* Introduced lambda functions, which are small, anonymous functions.
* Lambda functions use the syntax lambda arguments: expression.
* They are often used for simple, one-line operations and are first-class objects.
* Lambda functions can be used with higher-order functions like map and filter.

**6. Docstrings and Coding Style:**

* Covered best practices for writing docstrings, including concise summaries.
* Introduced function annotations as optional metadata about types.
* Emphasized Python coding style, including indentation, line length, and naming conventions.

**7. Lambda Function Example - Logical Operation:**

* Demonstrated a lambda function to perform a logical operation on user-input values x and y.
* Converted the result to a human-friendly "True" or "False" output.

**8. Example - Animal Information Output:**

* Took user input for animal type, species, and water consumption.
* Formatted the water consumption to display two decimal places.
* Used an f-string to display a user-friendly output with the entered information.

[**Operators-in-Python-Day-4-wee1**](https://github.com/andisiwenonkwenkwe/Operators-in-Python-Day-4-wee1#operators-in-python-day-4-wee1)

Operators in programming are symbols or keywords that allow you to perform operations or manipulations on data or variables. They are essential for creating expressions and statements in code, enabling you to perform tasks like arithmetic calculations, comparisons, logical operations, and more. Here are some common categories of operators in programming:

[**Arithmetic Operators:**](https://github.com/andisiwenonkwenkwe/Operators-in-Python-Day-4-wee1#arithmetic-operators)

* (Addition): Adds two values together.
* (Subtraction): Subtracts the right operand from the left operand.
* (Multiplication): Multiplies two values. / (Division): Divides the left operand by the right operand. % (Modulus): Returns the remainder of a division. \*\* (Exponentiation): Raises the left operand to the power of the right operand. // (Floor Division): Divides and returns the integer part of the quotient.

[**Comparison Operators (Relational Operators):**](https://github.com/andisiwenonkwenkwe/Operators-in-Python-Day-4-wee1#comparison-operators-relational-operators)

== (Equal to): Checks if two values are equal. != (Not equal to): Checks if two values are not equal. < (Less than): Checks if the left value is less than the right value.

(Greater than): Checks if the left value is greater than the right value. <= (Less than or equal to): Checks if the left value is less than or equal to the right value. = (Greater than or equal to): Checks if the left value is greater than or equal to the right value.

[**Logical Operators:**](https://github.com/andisiwenonkwenkwe/Operators-in-Python-Day-4-wee1#logical-operators)

and (Logical AND): Returns True if both conditions are True. or (Logical OR): Returns True if at least one condition is True. not (Logical NOT): Returns the opposite of the condition.

[**Assignment Operators:**](https://github.com/andisiwenonkwenkwe/Operators-in-Python-Day-4-wee1#assignment-operators)

= (Assignment): Assigns a value to a variable. += (Add and Assign): Adds the right operand to the left operand and assigns the result to the left operand. -= (Subtract and Assign): Subtracts the right operand from the left operand and assigns the result to the left operand. \*= (Multiply and Assign): Multiplies the left operand by the right operand and assigns the result to the left operand. /= (Divide and Assign): Divides the left operand by the right operand and assigns the result to the left operand. %= (Modulus and Assign): Performs modulus on the left operand with the right operand and assigns the result to the left operand.

[**Bitwise Operators:**](https://github.com/andisiwenonkwenkwe/Operators-in-Python-Day-4-wee1#bitwise-operators)

& (Bitwise AND) | (Bitwise OR) ^ (Bitwise XOR) ~ (Bitwise NOT) << (Left Shift)

(Right Shift)

[**Membership Operators:**](https://github.com/andisiwenonkwenkwe/Operators-in-Python-Day-4-wee1#membership-operators)

in: Checks if a value exists in a sequence (e.g., in a list or string). not in: Checks if a value does not exist in a sequence.

[**Identity Operators:**](https://github.com/andisiwenonkwenkwe/Operators-in-Python-Day-4-wee1#identity-operators)

is: Checks if two objects have the same identity. is not: Checks if two objects do not have the same identity. These operators are essential for building expressions and statements that manipulate data and make decisions in your code. The choice of operators depends on the specific tasks you need to perform in your programming language. **Operators concept:**

* Operators in programming are used for testing conditions, manipulating values, and performing various operations.
* Expressions are combinations of values, variables, and operators that can be evaluated to produce a result.
* Comparing different types depends on the programming language and its type coercion rules. Incompatible types can result in exceptions, like TypeError.
* Complex numbers, which include real and imaginary parts, may raise exceptions when compared to other incompatible data types due to a lack of natural ordering.
* Ordering operators like <, <=, >, and >= are used to compare values, but not all data types can be compared. They work with data types that have a defined order.

**Operators and Functions:**

* Mathematical operators include +, -, \*, /, //, %, -a, and +a for performing arithmetic operations.
* Mathematical functions like abs(), int(), float(), complex(), divmod(), and pow() perform specific mathematical tasks.
* Additional mathematical functions like math.trunc(), round(), math.floor(), and math.ceil() provide rounding and truncation capabilities.
* Comparison operators include <, <=, >, >=, ==, !=, is, and is not for comparing values and object identity.

**Daily Notes - Activity 1:**

* This activity calculates the number of water bottles required to hold a given amount of water in liters.
* It takes user input for the amount of water, calculates the number of bottles, and the remaining water in liters.
* Results are displayed with proper formatting.

**Daily Notes - Activity 2:**

* This activity reports the amount of water released and the amount of water remaining in liters, converting milliliters to liters for display.
* It also calculates and displays the percentage of water remaining.
* Results are presented in a report format with appropriate formatting and headers.