Download Your Daily Notes

Daily Notes - Python in the job market

it is nice to know python

Daily Notes - Activity 1 - Installing Anaconda on Windows

i managed to download the anaconda

Daily Notes - History of Python

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.

Daily Notes - Invoking the Interpreter

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.

Daily Notes - Activity 2 - Interactive Mode

the download steps were easy

Daily Notes - Comments in Python

comments are a programming language construct used to insert human-readable text in the source code of a program.

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.

Daily Notes - Activity 3 - Using Python

there is more than one way to print a word using stars in python, i enjoyed exploring them.

i understand that the trick is patterns.

Daily Notes - Activity 4 - Strings

No answer yet.

My Views on the Day

- being able to write/ draw something in Python using stars.

- knowing how to comment and how to use escape sequences.

-installing Anaconda and Python Notebook.

-using stars.

Daily Notes - Day 1 Reflections

day 1 was great

My own views on Introduction to 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.

Every variable is created with an initial value. A variable can be in three states:

Variable creation (Declaration)
Variable assignment (Initialization)
Variable changed (Execution)

ii python, variables are defined using =

Daily Notes - Introduction to Variables

Here are a few examples of valid variable names:

c
ref_number
admin
aVeryLongName
Here are a few examples of invalid variable names:

True
$name
12Graph

In Python identifiers are case sensitive, A second rule is that variables cannot have the same name as Python's keywords.

The dir() function in Python is a built-in function that is used to get a list of all attributes and methods associated with an object. When you call dir() with no arguments, it returns a list of names in the current local scope. However, when you call dir() with an object as an argument, it returns a list of names of attributes and methods associated with that object.

To understand how to use dir() with the __builtins__ attribute, let's break it down:

__builtins__:

In Python, __builtins__ is a module that contains the built-in functions, exceptions, and attributes of Python. It's automatically available in every Python script or session and provides a set of fundamental functions and objects that are always accessible without the need to import any modules.
dir(__builtins__):

When you call dir() with __builtins__ as an argument, you are essentially asking Python to provide a list of all the built-in attributes, functions, and objects that are available in the standard Python environment. This can include things like built-in functions (e.g., print(), len()), exceptions (e.g., ValueError, TypeError), and various constants (e.g., True, False, None).

Daily Notes - Using variables

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).

Daily Notes - Casting

Casting can be done in two ways:

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: 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.

Daily Notes - Defining Functions

No answer yet.

Daily Notes - Activity 1 - Consolidating Learnings

1. String are chracters
2. intigers are numbers
3. int()
4.str()
5. Variable names are in lowercase with words separated by underscores (snake_case).
Function names use snake_case.
Class names use CamelCase (CapWords).
Module names are in lowercase.
Constants are typically in uppercase.

Private names may start with a single underscore (e.g., _my_variable).
Dunder (double underscore) names are used for special methods or attributes (e.g., __init__).
Package names are in lowercase.
Acronyms with two or fewer characters are in uppercase.
Consistent spacing and indentation are used for readability.

My Views on the Day

- casting
-casting
-using variables
-none

Daily Notes - Day 1 Reflections

great day of elarning.

My own views on Data Types

it is important to know which data type to choose in order for your program to run correctly on a computer.

some of the available data types include:
1. Integers - These represent numbers in an unlimited range. This is only limited by a machine's memory.
2. Booleans - Evaluate to 'True or False', 1 or 0 respectively.
3. Floating point numbers - Floating-point numbers represent double-precision numbers.
4. Complex numbers - Complex numbers represent numbers as a pair of double-precision numbers.
5. Strings - A sequence of Unicode characters e.g. a word or a sentence that can be manipulated.

Daily Notes - Data Types

Literals are an alternative to using variables.

Examples of literals include:

"This is only a string"
"\t"
2

literals are values that never change (they are a constant).

Daily Notes - Integers

Python offers three distinctive numeric types to cater to different needs in programming:

Integers: Integers represent whole numbers, which can be both positive and negative. They don't have decimal parts and are used for a wide range of numerical operations. The range of an integer is determined by the available memory on the machine, making them suitable for various applications.

Static Typing: Python is considered a statically typed language because variable types are assigned at the time of declaration, and once assigned, a variable's type cannot be changed. This helps ensure data type consistency and allows for efficient program execution.

Integer Manipulation: Python provides basic arithmetic operations for manipulating integers:

Addition (a + b): Adds two integers.
Subtraction (a - b): Subtracts one integer from another.
Multiplication (a * b): Multiplies two integers.
Division (a / b): Divides one integer by another, always returning a floating-point number. This means that division can result in a decimal or floating-point value even if the inputs are integers.

Python's ability to differentiate between these numeric types allows programmers to choose the most appropriate data type for their specific needs, ensuring both accuracy and performance efficiency in their programs.

The plus '+' operator can be used universally, which means that you can use it for purposes other than to adding numbers together. The plus operator can also be used to concatenate strings.

The provided example is a Python program for a company, The program calculates the total cost of vegetables purchased by a store, taking into account fixed prices and special discounts.

In the provided example, the user inputs quantities for each vegetable, and the program calculates the total cost accordingly.

Daily Notes - Floating point numbers

The provided example demonstrates different ways to format floating-point numbers in Python using the % operator. It focuses on controlling the precision and formatting of floating-point values.

The examples illustrate how to format floating-point numbers, control their precision, and include signs for both positive and negative values. Formatting is achieved using format specifiers within the % operator, allowing for a customized display of numeric data.

Daily Notes - Unpacking Argument Lists

The provided examples illustrate various aspects of working with strings in Python, as well as input handling and code formatting. Here's a summary of the key points from these examples:

Example 6: Converting to String

The program prompts the user to enter their name, surname, and current age.
The input() function reads user input, and str() is used to convert the input into strings.
The program then combines these inputs to create a sentence.
The final sentence is printed, displaying the user's name, surname, and age.

Example 7: Using the += Operator with Strings

The += operator is used to append or concatenate strings together, making it easy to build longer strings incrementally.
Example 8: Using the End of Line Escape Sequence

The backslash \ is used to break long lines of code into multiple lines for better readability without affecting the string.

Example 9: Enclosing the Expression in Brackets

To enhance code readability, you can enclose an expression in parentheses and span it over multiple lines.
These examples demonstrate different techniques for working with strings, taking user input, and formatting code for better clarity and organization. Strings are a fundamental part of Python and are used extensively for text processing and data manipulation.

Daily Notes - Lambda Expressions

Lambda expressions in Python are a way to create small, anonymous functions. They are often used for simple operations where defining a full function with a name and the def keyword is not necessary.

Daily Notes - Conventions about the content and formatting of documentation strings

Documentation Strings (Docstrings):

The first line of a docstring should be a concise summary of an object's purpose.
Use a blank line to separate the summary from the rest of the description.
Follow a specific indentation convention within docstrings.
Function Annotations:

Annotations provide optional type information for function parameters and return values.
Annotations are stored in the __annotations__ attribute.
Parameters are annotated using colons, and return values with ->.

Coding Style (PEP 8):

PEP 8 is the widely-adopted coding style guide for Python.
It promotes 4-space indentation, limiting lines to 79 characters, using blank lines to separate code blocks, and putting comments on separate lines.
Consistent naming conventions and careful handling of encodings are encouraged.

Activity 1 - Consolidating Learnings

The OR operator, denoted as or, returns True if at least one of the operands is True. It evaluates to False only when both operands are False.

The NOT operator, denoted as not, returns the opposite of the operand's truth value. If the operand is True, not returns False, and if the operand is False, not returns True.

These operators are often used in logical expressions to make decisions and perform conditional operations in Python.

My Views on the Day

important features of the day included:

Learning about data types in Python, including integers, booleans, and complex numbers.
Exploring the use of mathematical operators for calculations in Python.
Understanding how to work with strings, including string formatting and concatenation.
Learning about lambda expressions and their applications.
Covering documentation strings (docstrings) and coding style conventions in Python.
Beneficial activities included:

Exploring Python's data types and mathematical operators, which are fundamental concepts in programming.
Writing Python programs to perform calculations and string manipulations, reinforcing practical coding skills.
Understanding the use of lambda expressions for creating small, anonymous functions.
Learning about documentation strings and coding style conventions to write more readable and maintainable code.
Activities that were too easy may vary depending on the individual's prior experience and knowledge of Python. For some, the introductory concepts may have been straightforward, while others may have found certain topics more challenging.

Activities that may need more time could include:

Complex numbers: Exploring more advanced applications and mathematical operations involving complex numbers.
Further practice with lambda expressions: Creating more complex and practical lambda functions.
Advanced string manipulation: Learning about regular expressions, advanced string formatting, and text processing.
Coding style and documentation: Delving deeper into best practices for writing clean and well-documented Python code.

Daily Notes - Day 3 Reflections

the need for more activities depends

Activity 2

```
# Get input values for animalType, animalSpecies, and waterConsumption
animalType = input("Enter the animal's type: ")
animalSpecies = input("Enter the animal's species: ")
```

```
waterConsumption = input("Enter how many liters of water it drinks a day: ")

# Convert waterConsumption to a float
waterConsumption = float(waterConsumption)

# Create and print the message
message = f"The {animalType} ({animalSpecies}) drinks {waterConsumption} liters of water a day."
print(message)
```

Boolean data types in Python represent logical values, and they have corresponding integer values: True (1) or False (0). When used as strings, they are "True" and "False," not "1" and "0." It's important to note that True and False are case-sensitive in Python.

Boolean data types are primarily used to test conditions to determine their validity. There are three main logical operators used for testing conditions between two arguments:

1. **and-operator**: Returns True if both conditions are True, otherwise False.
2. **or-operator**: Returns True if at least one of the conditions is True, otherwise False.
3. **not-operator**: Returns the opposite of the condition; if the condition is True, not returns False, and vice versa.

In Python, certain values are considered False, including:

- False
- None
- Zero for any numeric data type (0, 0.0, 0j)
- An empty sequence or mapping (e.g., empty strings, empty lists, empty dictionaries)
- Instances of user-defined classes that define a **__bool__()** method returning zero or False.

All other values are considered True. This means that many objects will evaluate to True.

Operators and built-in functions that return Boolean results always return either False (or 0) or True (or 1). The Boolean "or" and "and" operations always return one of these options, either True or False, based on the conditions being tested.

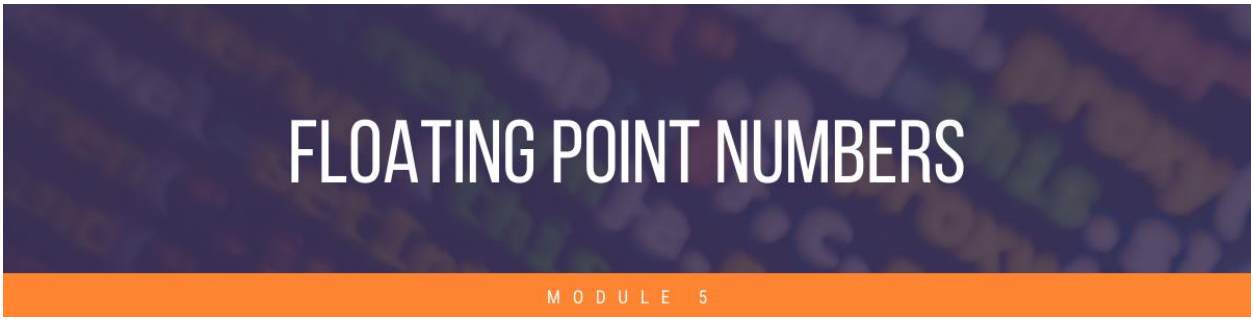
Boolean manipulation in Python involves the use of logical operators to work with Boolean values, which can be either True or False. Here is a summary of common Boolean manipulation operators:

1. **or**: The **or** operator returns True if either of the two conditions is True. If both conditions are False, it returns False. It functions like an "either-or" operation.
2. **and**: The **and** operator returns True only if both conditions are True. If at least one of the conditions is False, it returns False. It functions like a logical "and" operation.
3. **not**: The **not** operator is used to invert the value of a Boolean expression. If the expression is True, **not** returns False, and if the expression is False, **not** returns True.

In electronic circuits, diodes work on similar principles, where they can be in either an "on" (True) or "off" (False) state. For example:

- **AND Gate**: In the AND gate, both inputs must be "on" (True) for the gate to produce an "on" (True) output. If one or both inputs are "off" (False), the output is "off" (False).
- **NOT Gate**: The NOT gate inverts the input. If the input is "on" (True), it produces an "off" (False) output, and if the input is "off" (False), it produces an "on" (True) output.

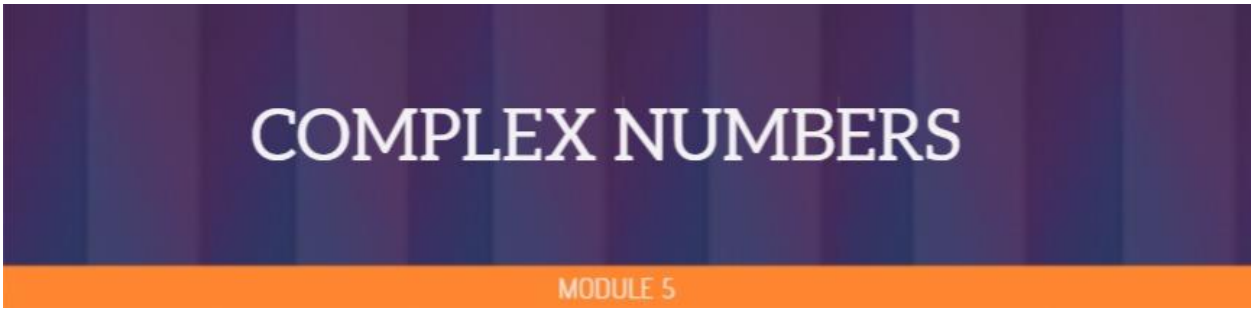
These principles are fundamental in digital logic and are also used in programming to make decisions based on the truth values of different conditions.



Better known as floats.

Float is the data type that manages numbers with decimal places with very accurate precision. The float data type can be called as a function with zero or 1 argument of any data type. If no argument is given, then float returns 0.0. If an argument is given, an attempt will then be made to convert the value to a float data type, but this does not mean it is always possible.

A string value cast to a float must contain only numbers and only one occurrence of the dot (.) character.

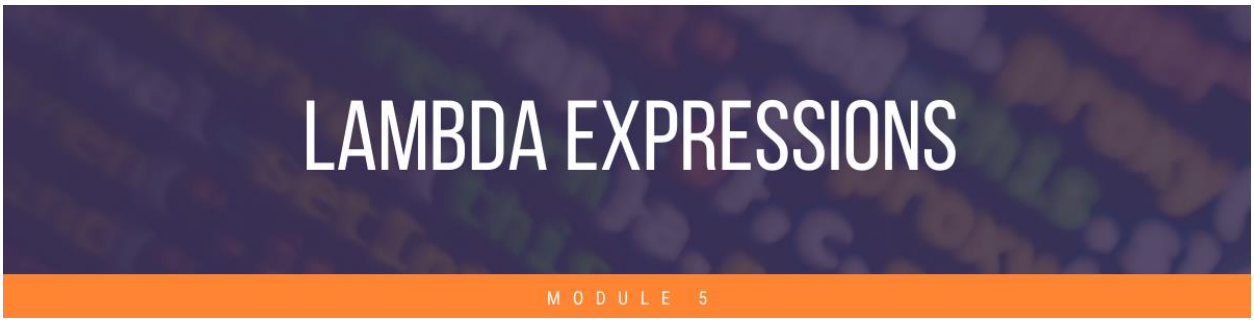


Complex numbers in Python are represented as a combination of two parts: a real part (a floating-point number) and an imaginary part (also a floating-point number). The imaginary part is expressed with a suffix "j" (or "J"). Complex numbers are natively supported in Python and are used to combine two values into a single manageable entity, commonly found in scientific and engineering contexts.

Strings in Python are represented by the **str** data type, which is immutable, meaning their content cannot be changed after creation. They are sequences of Unicode characters, allowing the formation of single, manageable strings of text. Key points about strings include:

1. **Creation of Strings**:
    - Strings can be created by using the **str()** constructor, which returns an empty string if no argument is provided.
    - When an argument other than a string value is passed to the **str()** constructor, it converts the argument into its string representation. For example, **str(17.2354)** is the same as **str("17.2354")**, where the floating-point number is converted into a string.
2. **String Immutability**:
    - Strings are immutable, which means their content cannot be modified once created. Any operation that appears to change a string actually creates a new string.
3. **Converting Other Data Types to Strings**:
    - The **str()** function is often used to convert data of other types into strings, allowing for easy representation of various data in text form.

Strings are fundamental in Python and are extensively used for handling text, processing data, and working with textual information in various applications.



Lambda expressions in Python are a way to create small, anonymous functions. They are often used for simple operations where defining a full function with a name and the **def** keyword is not necessary. Here's a summary of key points regarding lambda expressions:

- A lambda expression is defined using the **lambda** keyword followed by the arguments and a colon, and then the expression to be evaluated.
- Lambda functions are generally limited to a single expression, meaning they can't contain multiple statements or complex logic.
- Lambda functions are often used where function objects are required, such as when passing a function as an argument to another function.
- Lambda functions can access variables from their containing scope, similar to nested function definitions.

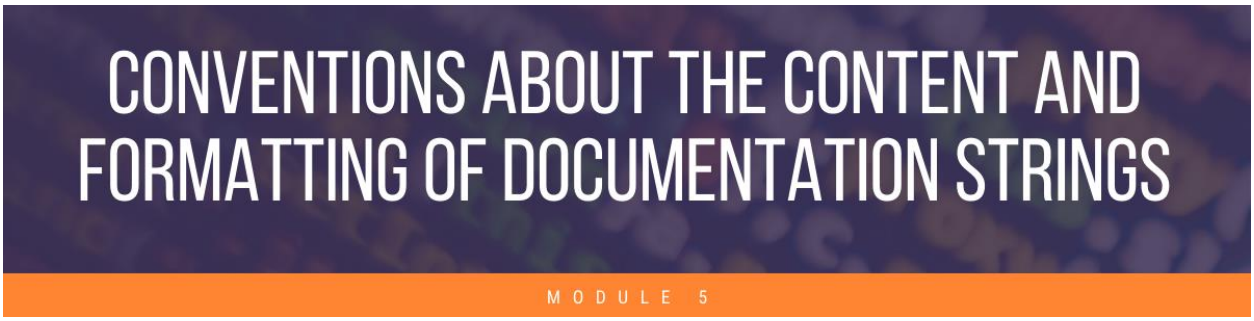
Example 1: Creating a Lambda Function

- In this example, a lambda function is defined as **lambda a, b: a + b**, which takes two arguments and returns their sum.

Example 2: Using a Lambda Function

- This example demonstrates the use of a lambda function to return a function (**make_incrementor**) that increments a value by a given amount.
- Another example shows how lambda functions can be used to sort a list of pairs based on the second element of each pair.

Lambda expressions provide a concise way to create simple, single-expression functions, making code more readable and efficient for certain use cases.



Documentation Strings (Docstrings):

- The first line of a docstring should be a concise summary of the object's purpose, typically not explicitly stating the object's name or type. It should start with a capital letter and end with a period.
- If the docstring has multiple lines, the second line should be blank to separate the summary from the rest of the description.
- The indentation within a docstring should be stripped, following a specific convention.
- An example of a multi-line docstring is provided for illustration.

Function Annotations:

- Function annotations are optional metadata about the types used by user-defined functions.
- Annotations are stored in the **__annotations__** attribute of the function as a dictionary and do not affect the function's behavior.
- Parameter annotations are defined by a colon after the parameter name, followed by an expression evaluating the value of the annotation.
- Return annotations are defined by **->** followed by an expression, between the parameter list and the colon.
- An example function is shown with parameter and return value annotations.

Coding Style (PEP 8):

- Coding style is important for readability and should be consistent across projects.
- Python Enhancement Proposal 8 (PEP 8) is the style guide most projects adhere to, promoting readable and eye-pleasing coding style.
- Key points from PEP 8 include using 4-space indentation (no tabs), limiting lines to 79 characters, using blank lines to separate functions and classes, and putting comments on a line of their own.
- Docstrings are encouraged for documenting functions and modules.
- Naming conventions are provided, such as using UpperCamelCase for classes and lowercase_with_underscores for functions and methods.

- Consistency in using self as the name for the first method argument in classes is recommended.
- Encoding and non-ASCII characters should be handled carefully for international environments.

Following these conventions and PEP 8 guidelines can improve the readability and maintainability of Python code, making it easier for others to understand and work with.