Python

Introduction to variables

* Variables are a temporal storage space in a computer’s memory.
* It acts as a container to hold a different number of data items or values.
* They are also used to move data from different functions.
* In python variables are defined in a standard way by using the assignment character.
* This changes the value of the variable.

Rules in naming the variables.

* Variable names may contain any upper- or lower-case letters a number or underscore characters.
* They may not begin with a number or contain spaces.
* Variables cannot have the same name as pythons’ keywords(dir.).

Using variables

* All variables must be assigned to a data type like a string or an integer.
* In python variables are assigned automatically to an appropriate data type.

Casting

* Casting can be done in two ways, that’s implicitly and explicitly.
* Implicitly The compiler automatically casts a value from one data type to another when assured that there will be no data loss.
* Explicitly A value cannot be automatically cast from one data type to another if it will result in data loss. Extra code must be written to ensure that the value stays the same and only the data type changes.

Datatype

* 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

Python's numeric types (integers, floats, and complex numbers) cater to various numerical needs, from whole numbers to real and complex values. The language's static typing ensures data consistency, and the range of integers is primarily limited by available system memory, allowing for flexibility in handling different numeric values.

Operators

In programming, there are four fundamental operators for performing calculations: addition (+), subtraction (-), multiplication (\*), and division (/). These operators are used to manipulate numbers and perform arithmetic operations. Additionally, the plus operator (+) has universal applications, and it can be used not only for numerical addition but also for string concatenation.

Booleans

the Boolean data type in Python represents two values: True (which corresponds to 1) and False (which corresponds to 0). These values are used for making logical decisions in your programs.

Python's Boolean data type also treats "True" and "False" as the string representations of True and False, respectively. Keep in mind that these are case-sensitive.

To test conditions between two values, you can use three logical operators:

1. and: Returns True if both conditions are True.
2. or: Returns True if at least one of the conditions is True.
3. not: Inverts the value of a condition (True becomes False, and vice versa).

Values that are considered as False in Python include False, None, 0 (for any numeric data type), empty sequences or mappings (e.g., empty lists, tuples, and dictionaries), and instances of user-defined classes where a special \_\_bool\_\_() method returns zero or False. Everything else is considered True.

Operators and built-in functions that produce Boolean results will always return either False (or 0) or True (or 1). The Boolean "or" and "and" operations will always return one of these two options: True or False. These concepts are fundamental for making decisions and controlling the flow of your programs based on conditions.

Floating point numbers

In Python, floating-point numbers, often referred to as "floats," are a data type used to represent numbers with decimal places. Floats offer high precision for handling such numbers. They can be used as a function with zero or one argument of any data type:

* If no argument is given, the float () function returns 0.0.
* If an argument is provided, the function attempts to convert the value to a floating-point number. However, not all conversions are guaranteed to be successful. For instance:
  + float ("21.765") will successfully convert the string to a float.
  + float("FF909A") will raise an exception because it cannot be converted to a valid float, as it contains non-numeric characters.

When casting a string to a float, the string must consist of numerical characters and can contain only one occurrence of the dot (.) character, which represents the decimal point.

Floats in Python are used to manage numbers with decimal places and offer precise representation. The float () function can convert values to floats, but it's important to ensure that the input is valid for the conversion to succeed, such as having only numeric characters and a single decimal point.

Complex numbers

Complex numbers in Python are a way to represent two numbers in a single variable. Each complex number consists of two parts:

1. The real part, which is a floating-point number (a decimal number).

2. The imaginary part, which is also a floating-point number but is multiplied by the imaginary unit (i or j), and it represents the square root of -1.

In Python, you can create complex numbers using the `complex ()` function, which takes two arguments: the real part and the imaginary part. For example, `complex (4, 8)` represents a complex number with a real part of 4 and an imaginary part of 8. The imaginary part is often denoted with a "j" or "J" suffix, so you can also write this as `4 + 8j`.

Complex numbers are used in Python to combine two values into a single, manageable number. They are particularly useful for mathematical and scientific calculations where real and imaginary components need to be represented together. For example, complex numbers are commonly used in electrical engineering and physics to model and analyse alternating current (AC) circuits and quantum mechanics.

Strings

rings are a sequence of Unicode characters which form a single manageable string. The str data type can be called to create a string; when there is no argument supplied, it returns an empty string. s = str ("") is the same as s = str (), when an argument is passed to the string method that is not a string value, it is passed as a string representation of the type supplied: s = str(17.2354), is the same as s = str("17.2354"). The string function is often used to convert other data types to strings.

Lambda expressions

* In Python, lambda expressions are a way to create small, anonymous functions.
* They are defined using the lambda keyword and are often used when you need a simple function for a short period.
* Lambda expressions can take any number of arguments but are restricted to a single expression.
* They are essentially syntactic sugar for defining small functions.

Conventions about the content and formatting of documentation strings

The first line should always be a short

 It should not explicitly state the object’s name or type.

Operators

Operators are used to testing conditions and manipulating values: This is correct. In Python, operators are used to perform various operations on values, such as arithmetic operations, comparison operations, logical operations, and more.

Most statements contain expressions: This is also correct. Expressions are fundamental building blocks of Python statements. An expression is a combination of values and operators that can be evaluated to produce a result.

An example of an expression is: 2 + 3: Absolutely correct. 2 + 3 is an expression that represents the addition of the numbers 2 and 3.

When two objects of a different type, like str and int, are compared they are never equal: This is true for most cases. In Python, objects of different types are typically not considered equal, and attempting to compare them directly may result in False. However, some implicit type conversions may occur in certain situations. For example, if you explicitly convert a string to an integer using int(), you can compare them.

Except for different numeric types like int and float which can be equal: That's correct. In Python, numeric types like int and float can be compared, and they are considered equal if their values are the same, even if one is an integer and the other is a floating-point number.

The <, <=, >, and >= operators raise a TypeError exception when any operand is a complex number: This is not entirely accurate. Python does support comparisons between complex numbers and other numeric types like integers and floats using these operators. However, comparing complex numbers to non-numeric types may result in a TypeError because there is no defined ordering between complex numbers and most non-numeric types