**Module 3. Understanding Data Types**

**Learning Objectives**

* Distinguish between different data types in Python, including numeric types (int, float, complex), text types (str), and Boolean types (bool), and recognize the appropriate usage and characteristics of each data type.
* Obtain proficiency in handling basic arithmetic operations with different numeric data types and use the type() function to check and confirm the data types of various variables and expressions in Python.
* Understand string operations, including concatenation, slicing, and formatting, and learn to work with single and double quotes, as well as utilize multi-line strings and escape characters.
* Convert between different data types using Python’s built-in functions such as int(), float(), and str(), and handle user input by converting it to the required data type for further processing.
* Use Boolean expressions and operators for decision-making in programs, and understand the application of comparison operators and logical operators (and, or, not) in control flow statements.

# **Numeric Data Types**

Data types are fundamental in programming, as they define the kind of data variables can store and the operations they support. Python includes default built-in data types grouped into several categories. These types are used to store different kinds of data. This module will explore commonly used simple data types, including text types, numeric types, and Boolean types.

|  |  |
| --- | --- |
| **Data Types** | **Keywords** |
| Text Type | **str** |
| Numeric Types | **int, float, complex** |
| Sequence Types | **list, tuple, range** |
| Mapping Type | **dict** |
| Set Types | **set, frozenset** |
| Boolean Type | **bool** |
| Binary Types | **bytes, bytearray, memoryview** |
| None Type | **NoneType** |

When computers were initially created, their primary role was to perform complex calculations, and this remains a significant use today. Problems that involve mathematical formulas are straightforward to convert into Python programs. In this section, we will delve deeper into programs specifically designed for numerical computations.

Numbers in Python are divided into three main types: integers, floating-point numbers, and complex numbers. Integers are whole numbers without a fractional component, while floating-point numbers contain a decimal point, making them suitable for representing more precise values. Complex numbers consist of a real part and an imaginary part, denoted by the suffix 'j'. Python provides a wide range of operations for working with these number types, including arithmetic operations, type conversion, and built-in functions for mathematical calculations.

Numeric variables are created when you assign a value to them. To check the type of any object in Python, you can use the ***type()*** function.

|  |
| --- |
| x = 10 # int  y = 3.14 # float  z = 1j # complex  print(type(x))  print(type(y))  print(type(z)) |

***Integer***

Integers (***int***) in Python refer to whole numbers that can be positive or negative, without decimal points, and can be of unlimited size.

|  |
| --- |
| x = 0 # zero  y = 42233435353211211 # positive integer  z = -17 # negative integer  print(type(x))  print(type(y))  print(type(z)) |

***Float***

A float, or "floating point number," in Python is a number that can be positive or negative and contains one or more decimals.

|  |
| --- |
| x = 0.5  y = -10.5  z = 3.14  print(type(x))  print(type(y))  print(type(z)) |

The following program demonstrates basic input handling, arithmetic operations, and output formatting for calculating and displaying the sum and average of three test scores.

|  |
| --- |
| # test\_scores.py  # A program to calculate the sum and average of three test scores  def main():  print("Test Scores Calculator")  print()    # Prompting the user to enter three test scores  score1 = float(input("Enter score 1: "))  score2 = float(input("Enter score 2: "))  score3 = float(input("Enter score 3: "))    # Calculating the sum of the scores  total = score1 + score2 + score3    # Calculating the average of the scores  average = total / 3    print()  print("Sum of the scores:", total)  print("Average of the scores:", average)  # Calling the main function to execute the program  main() |

Example output:

|  |
| --- |
| Test Scores Calculator  Enter score 1: 85  Enter score 2: 92  Enter score 3: 78  Sum of the scores: 255.0  Average of the scores: 85.0 |

In the program, the *main()* function prompts the user to enter three test scores (*score1*, *score2*, and *score3*) using *float(input())* to handle decimal input. Then, it calculates the sum of the three scores using *total = score1 + score2 + score3*. In addition, it calculates the average of the three scores by dividing the total sum by 3 (*average = total / 3*). Finally, it prints out the calculated sum and average of the scores.

***Type Conversion***

In Python, you can convert between different data types using *int()* and *float()* methods. The *int()* function is used to convert a number or a string containing a numeric value to an integer.

|  |
| --- |
| num\_float = 3.75  num\_int = int(num\_float)  print(num\_int) # Output: 3 |

The *float()* function converts a number or a string containing a numeric value to a float.

|  |
| --- |
| num\_int = 5  num\_float = float(num\_int)  print(num\_float) # Output: 5.0 |

These methods are handy when you need to change the type of a variable or handle different types of numeric inputs in your Python programs.

***Random Numbers***

Python does not have a built-in function to make a random number, but Python has a built-in module called ***random*** that can be used to make random numbers. You need to import the random module to generate random numbers. The following code generates and displays a random number between 1 and 6.

|  |
| --- |
| import random  print(random.randrange(1, 7)) |

# **Strings**

In Python, strings can indeed be enclosed in either single (' ') or double (" ") quotation marks, and both are functionally equivalent. This flexibility allows you to choose the style that best fits your needs, especially when dealing with strings that contain quotation marks.

Here are a few examples to demonstrate different usages:

|  |
| --- |
| print("Hello") #Output: Hello  print('Hello') #Output: Hello |

If your string contains a single quote, you can use double quotes to avoid needing to escape the single quote, and vice versa.

|  |
| --- |
| print("It's a beautiful day!")  print('He said, "Python is great!"') |

If you need to use the same type of quotation mark inside the string, you can escape it with a backslash (\):

|  |
| --- |
| print('It\'s a beautiful day!')  print("He said, \"Python is great!\"") |

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

|  |
| --- |
| a = "Hello World"  print(a) |

***Multiline Strings***

Multi-line strings in Python can be defined using triple quotes, either """ or '''. These are particularly useful for strings that span multiple lines, such as long text blocks, code snippets, or formatted text.

|  |
| --- |
| a = """Lorem ipsum dolor sit amet,  consectetur adipiscing elit,  sed do eiusmod tempor incididunt  ut labore et dolore magna aliqua."""  print(a) |

***Slicing Strings***

Slicing is a powerful feature in Python that allows you to extract a subset of characters from a string using a specified range of indices. The general syntax for slicing is ***string[start:end]***, where ***start*** is the index of the first character you want to include, and ***end*** is the index of the character you want to stop before (not included in the slice).

Here's an example that demonstrates how to slice a string to get characters from position 2 to 5 (not included):

|  |
| --- |
| b = "Hello, World!"  print(b[2:5]) #Output: llo |

The syntax for slicing can be broken down into three parts:

* ***start*** (optional): The starting index of the slice. The character at this index is included in the slice. If omitted, the slice starts from the beginning of the string.
* ***end*** (optional): The ending index of the slice. The character at this index is not included in the slice. If omitted, the slice goes until the end of the string.
* ***step*** (optional): The step size, or the interval between indices. If omitted, the default step is 1.

|  |
| --- |
| b = "Hello, World!"  # From index 2 to 5 (5 not included)  print(b[2:5]) # Output: llo  # From start to index 5  print(b[:5]) # Output: Hello  # From index 2 to end  print(b[2:]) # Output: llo, World! |

You can also specify a step size to skip characters:

|  |
| --- |
| b = "Hello, World!"  # Every second character from index 0 to 5  print(b[0:5:2]) # Output: Hlo  # Reverse the string  print(b[::-1]) # Output: !dlroW ,olleH |

Negative indices can be used to count from the end of the string:

|  |
| --- |
| b = "Hello, World!"  # From index -5 to -2 (not included)  print(b[-5:-2]) # Output: orl |

***String Concatenation***

String concatenation in Python allows you to combine two or more strings into one. This can be done using the ***+*** operator. Here are some examples to illustrate this concept:

To concatenate two strings, you simply use the ***+*** operator:

|  |
| --- |
| a = "Hello"  b = "World"  c = a + b  print(c) #Output: HelloWorld |

If you want to add a space between the concatenated strings, you include the space within the quotation marks:

|  |
| --- |
| a = "Hello"  b = "World"  c = a + " " + b  print(c) #Output: Hello World |

***String Format***

In Python, we cannot combine strings and numbers like this:

|  |
| --- |
| age = 25  txt = "Hello, my name is John, and I am " + age #ERROR  print(txt) |

However, strings and numbers can be combined by utilizing ***f***-strings or the ***format()*** method.

Using f-strings is an efficient and powerful way to format strings in Python. They allow you to embed expressions inside string literals using curly braces ***{}***. This makes it easy to combine strings and variables, perform calculations, and format values.

To create an f-string, prefix the string with the letter f and use curly braces {} to include variables or expressions:

|  |
| --- |
| age = 25  txt = f"Hello, my name is John, and I am {age}."  print(txt) #Output: Hello, my name is John, I am 25. |

You also can include variables directly within the curly braces:

|  |
| --- |
| price = 35  txt = f"The price is {price} dollars."  print(txt) #Output: The price is 35 dollars. |

The ***format()*** method provides another way to format strings, which is particularly useful for creating complex formatted strings. This method allows you to insert values into a string using placeholders defined by curly braces ***{}***.

The simplest use of the ***format()*** method involves placing curly braces ***{}*** within a string and passing the values you want to insert as arguments to the ***format()*** method.

|  |
| --- |
| name = "John"  age = 25  txt = "My name is {}, and I am {} years old.".format(name, age)  print(txt) #Output: My name is John, and I am 25 years old. |

You can use positional arguments by placing numbers inside the curly braces, indicating the position of the arguments.

|  |
| --- |
| name = "John"  age = 25  txt = "My name is {0}, and I am {1} years old. {0} is learning Python.".format(name, age)  print(txt) #Output: My name is John, and I am 36 years old. John is learning Python. |

# **Boolean**

In Python, Booleans are used to evaluate expressions and make decisions as same as in other programming languages. The Boolean data type has two possible values: ***True*** and ***False***.

When you compare two values, the expression is evaluated, and Python returns a Boolean value (True or False).

Python supports several comparison operators that return Boolean values:

|  |  |
| --- | --- |
| > | greater than |
| < | less than |
| == | equal to |
| != | not equal to |
| >= | greater than or equal to |
| <= | less than or equal to |

|  |
| --- |
| print(20 > 10) # True  print(20 == 10) # False  print(20 < 10) # False |

Booleans are often used in control flow statements like if statements to determine which block of code should be executed.

|  |
| --- |
| x = 55  y = 26  if y > x:  print("y is greater than x")  else:  print("y is not greater than x")  #Output: y is not greater than x |

The ***bool()*** function allows you to evaluate any value and returns ***True*** or ***False***. This can be particularly useful for checking the truthiness of values.

|  |
| --- |
| print(bool("Hello World")) # True  print(bool(25)) # True  print(bool("")) # False  print(bool(0)) # False |

Python also provides logical operators to combine Boolean expressions:

* ***and***: Returns ***True*** if both operands are true
* ***or***: Returns ***True*** if at least one operand is true
* ***not***: Returns ***True*** if the operand is false

|  |
| --- |
| x = True  y = False  print(x and y) # False  print(x or y) # True  print(not x) # False |

# **Type Conversion**

In Python, you can specify a type for a variable using casting, which involves converting a value from one type to another. Python, being an object-oriented language, uses classes to define data types, including its primitive types. Casting is performed using constructor functions like ***int()***, ***float()***, and ***str()***.

***Casting to Integer***

The ***int()*** function converts a value to an integer. This can be done from an integer literal, a float literal (by removing all decimals), or a string literal (provided the string represents a whole number).

|  |
| --- |
| # From an integer literal  x = int(5)  print(x) # Output: 5  # From a float literal (decimals are truncated)  y = int(5.99)  print(y) # Output: 5  # From a string literal  z = int("10")  print(z) # Output: 10  # From a float in string format  # This will raise a ValueError  try:  w = int("10.5")  except ValueError as e:  print(e) # Output: invalid literal for int() with base 10: '10.5' |

***Casting to Float***

The ***float()*** function converts a value to a float. This can be done from an integer literal, a float literal, or a string literal (provided the string represents a float or an integer).

|  |
| --- |
| # From an integer literal  x = float(5)  print(x) # Output: 5.0  # From a float literal  y = float(5.99)  print(y) # Output: 5.99  # From a string literal  z = float("10.5")  print(z) # Output: 10.5  # From an integer in string format  w = float("10")  print(w) # Output: 10.0 |

***Casting to String***

The ***str()*** function converts a value to a string. This can be done from strings, integer literals, float literals, and many other data types.

|  |
| --- |
| # From a string literal  x = str("Hello")  print(x) # Output: Hello  # From an integer literal  y = str(10)  print(y) # Output: 10  # From a float literal  z = str(10.5)  print(z) # Output: 10.5  # From a boolean literal  w = str(True)  print(w) # Output: True |

***Practical Use Cases***

When reading input from a user, it is often read as a string. You might need to convert it to another type for further processing.

|  |
| --- |
| user\_input = input("Enter a number: ")  number = int(user\_input)  print(f"The number you entered is {number}") |

When dealing with numbers in string format, you need to convert them to the appropriate numeric type before performing calculations.

|  |
| --- |
| num1 = "10"  num2 = "20.5"  # Convert to appropriate numeric types  int\_num1 = int(num1)  float\_num2 = float(num2)  # Perform calculation  result = int\_num1 + float\_num2  print(result) # Output: 30.5 |

When you need to format numbers as strings for output purposes, you can use ***str()***.

|  |
| --- |
| age = 25  txt = "My name is John, and I am " + str(age) + " years old."  print(txt) # Output: My name is John, and I am 25 years old. |

# **Operators**

Operators are fundamental in programming as they enable you to perform various operations on variables and values. Python supports several types of operators, including arithmetic, comparison, logical, bitwise, assignment, identity, and membership operators.

***Arithmetic Operators***

Arithmetic operators are used to perform common mathematical operations.

|  |
| --- |
| x = 10  y = 5  # Addition  print(x + y) # Output: 15  # Subtraction  print(x - y) # Output: 5  # Multiplication  print(x \* y) # Output: 50  # Division  print(x / y) # Output: 2.0  # Modulus (remainder)  print(x % y) # Output: 0  # Exponentiation  print(x \*\* y) # Output: 100000  # Floor division  print(x // y) # Output: 2 |

***Comparison Operators***

Comparison operators compare two values and return a Boolean result (***True*** or ***False***).

|  |
| --- |
| x = 10  y = 5  # Equal to  print(x == y) # Output: False  # Not equal to  print(x != y) # Output: True  # Greater than  print(x > y) # Output: True  # Less than  print(x < y) # Output: False  # Greater than or equal to  print(x >= y) # Output: True  # Less than or equal to  print(x <= y) # Output: False |

***Logical Operators***

Logical operators are used to combine conditional statements.

|  |
| --- |
| x = True  y = False  # AND  print(x and y) # Output: False  # OR  print(x or y) # Output: True  # NOT  print(not x) # Output: False |

***Bitwise Operators***

Bitwise operators are used to perform bit-level operations on integers.

|  |
| --- |
| x = 10 # Binary: 1010  y = 4 # Binary: 0100  # AND  print(x & y) # Output: 0 (Binary: 0000)  # OR  print(x | y) # Output: 14 (Binary: 1110)  # XOR  print(x ^ y) # Output: 14 (Binary: 1110)  # NOT  print(~x) # Output: -11 (Binary: -1011)  # Left shift  print(x << 2) # Output: 40 (Binary: 101000)  # Right shift  print(x >> 2) # Output: 2 (Binary: 0010) |

***Assignment Operators***

Assignment operators are used to assign values to variables.

|  |
| --- |
| x = 10  # Assign  x = 5  print(x) # Output: 5  # Add and assign  x += 3  print(x) # Output: 8  # Subtract and assign  x -= 2  print(x) # Output: 6  # Multiply and assign  x \*= 2  print(x) # Output: 12  # Divide and assign  x /= 3  print(x) # Output: 4.0  # Modulus and assign  x %= 3  print(x) # Output: 1.0  # Exponentiation and assign  x \*\*= 2  print(x) # Output: 1.0  # Floor division and assign  x //= 1  print(x) # Output: 1.0 |

***Practical Examples***

You can combine different types of operators to create complex expressions.

|  |
| --- |
| x = 10  y = 5  z = 2  # Combine arithmetic and comparison operators  result = (x + y) > (z \* y)  print(result) # Output: True  # Combine logical operators  is\_valid = (x > y) and (y > z)  print(is\_valid) # Output: True |

Operators are often used in conditional statements to control the flow of the program.

|  |
| --- |
| age = 20  if age >= 18:  print("You are an adult.")  else:  print("You are a minor.")  #Output: You are an adult. |

**Summary**

1. Python supports various data types, including numeric types (int, float, complex), text types (str), sequence types (list, tuple, range), mapping types (dict), set types (set, frozenset), Boolean types (bool), binary types (bytes, bytearray, memoryview), and NoneType.
2. Numeric types in Python include integers (whole numbers without decimals), floats (numbers with decimals), and complex numbers (numbers with real and imaginary parts, denoted by 'j').
3. You can check the type of any object in Python using the ***type()*** function and convert between types using ***int()*** and ***float()*** functions.
4. A sample Python program can handle input, perform arithmetic operations, and format output to calculate and display the sum and average of test scores.
5. Python's random module allows generating random numbers, such as a random number between 1 and 6 using the ***random.randrange()*** function.
6. In Python, strings can be enclosed in single (' ') or double (" ") quotation marks, and multi-line strings can be defined using triple quotes (""" or ''') for text that spans multiple lines.
7. Slicing allows extraction of a substring using the syntax ***string[start:end:step]***, where indices can be positive or negative to count from the start or end of the string.
8. String concatenation combines strings using the + operator, and spaces can be added within the quotation marks for separation.
9. String formatting can be achieved using f-strings or the ***format()*** method, allowing the embedding of variables and expressions within strings.
10. Booleans in Python represent truth values (***True*** or ***False***) and are used with comparison operators to evaluate expressions and make decisions.
11. In Python, Booleans evaluate expressions and make decisions, with two possible values: True and False.
12. Comparison operators like >, <, ==, !=, >=, and <= return Boolean values when comparing two values.
13. Booleans are often used in control flow statements like if statements to determine which block of code to execute.
14. The ***bool()*** function can evaluate any value and return ***True*** or ***False***, which helps in checking the truthiness of values.
15. Python logical operators (***and***, ***or***, ***not***) combine Boolean expressions, allowing complex conditional statements.
16. Casting in Python involves converting a value from one type to another using functions like ***int()***, ***float()***, and ***str()***.
17. Arithmetic operators (+, -, \*, /, %, \*\*, //) perform basic mathematical operations, while comparison operators return Boolean results.
18. Logical operators (***and***, ***or***, ***not***) are used to combine conditional statements, and bitwise operators perform bit-level operations on integers.
19. Assignment operators (+=, -=, \*=, /=, %=, \*\*=, //=) assign values to variables and can perform operations simultaneously.
20. Combining different types of operators in complex expressions and conditional statements helps control the program's flow effectively.

**Programming Exercises**

**Exercise 1: Basic Numeric Operations**

Write a program that prompts the user to enter two numbers. Calculate and display the sum, difference, product, and quotient of these two numbers. Check and display the data types of the entered numbers and the results.

**Exercise 2: Random Number Generator**

Write a program that imports the random module. Generate and print five random integers between 1 and 100. Then, generate and print a random floating-point number between 0 and 1.

**Exercise 3: String Manipulation**

Write a program that prompts the user to enter their first name and last name. Concatenate the first name and last name into a full name and print it. Then, slice and print the first three letters of the first name and the last three letters of the last name.

**Exercise 4: Type Conversion**

Write a program that prompts the user to enter a floating-point number. Convert the floating-point number to an integer and print the result. Then, convert the integer back to a floating-point number and print the result.

**Exercise 5: Test Scores Calculator**

Write a program that prompts the user to enter three test scores. Calculate the sum and average of the test scores. Print the sum and average of the scores with appropriate labels.

**Exercise 6: String Handling and Quotation Marks**

Write a Python program that assigns the following text to a variable and prints it: He said, "Python's flexibility with strings is amazing!". Use both single and double quotation marks appropriately to avoid escape characters.

**Exercise 7: Multi-line Strings and Slicing**

Create a multi-line string variable with a short paragraph about your favorite hobby. Print the entire paragraph and then print a slice of the string that contains the first ten characters. Finally, print every second character from the slice.

**Exercise 8: String Concatenation**

Write a Python program that asks the user to input their first name and last name separately. Concatenate these inputs with a space in between to form their full name, and print the full name with the message: "Your full name is: <full\_name>".

**Exercise 9: String Formatting with f-strings**

Develop a Python program that prompts the user to enter their name, age, and favorite color. Use an f-string to format and print a message that says: "Hello, my name is <name>, I am <age> years old, and my favorite color is <color>."

**Exercise 10: Boolean Expressions and Comparison Operators**

Write a Python program that asks the user for the price of two different items. Compare the prices using comparison operators and print out which item is more expensive, or if they are priced the same. Use the format method to create the output message.