3. Data Types in Python

Connect with me: Youtube | LinkedIn | WhatsApp Channel | Web | Facebook | Twitter

- Download PDF
- To access the updated handouts, please click on the following link: https://yasirbhutta.github.io/python/docs/data-types.html
- Data Types in Python
    - 1. **Numeric Types:**
    - Understanding Dynamic Variables in Python with Examples
    - None
    - Type Hints
        - 1. **Type Hinting for Variables**
        - 2. **Type Hinting in Functions**
            - Example:
        - 3. **Type Hints for Collections**
            - Lists
            - Dictionaries
                - Example:
            - Sets
                - Example:
            - Functions with Collection Type Hints
                - Example:
    - Key Terms
    - Fix the error
    - True/False (Mark T for True and F for False)
    - Multiple Choice (Select the best answer)
    - Fill in the Blanks
    - Exercises
        - Exercise 1: Variable Assignment and Basic Operations
        - Exercise 2: Working with Different Data Types
        - Exercise 3: String Concatenation
        - Exercise 4: Boolean Operations
        - Exercise 5: Type Conversion
    - Review Questions
    - References and Bibliography

In Python, data types define the kind of value a variable can hold and the operations that can be performed on it. They act as blueprints, specifying how data is stored and manipulated in your programs.

Video: Variables in Python

3.1 **Numeric Types:**

```
- `int`: Stores whole (non-decimal) numbers, like `10`, `-5`, or `9999`.
- `float`: Represents floating-point numbers with decimals, like `3.14`, `-2.5e2` (scientific notation), or `1.2345678901234567` (limited precision).
- `complex`: Holds complex numbers with a real and imaginary part, like `3+2j` or `1.5-4.7j`.
```

- Example #1: How to use int variable
- Example #2: int variable
- Example #3: float variable

```
# Integer (int) to store age
age = 25

# Float (float) to store price with decimals
price = 14.99

# Complex number (complex) - not as common in everyday use
complex_num = 3 + 2j  # Imaginary unit represented by j
```

3.2 **String Type:**

```
- `str`: Represents textual data enclosed in single or double quotes, such as `"Hello, world!"`, `'This is a string'`, or multi-line strings using triple quotes (''' or """).
```

- Example #1: How to Convert a Python String to int
- Example #2: How to Convert a Python integer to string
- Example #3: Convert integer to octal and hexadecimal

```
# String (str) to store a name
name = "Alice"

# String with a sentence
greeting = "Hello, how are you?"

# Multi-line string using triple quotes
message = """This is a message
that spans multiple lines."""
```

3.3 **Boolean Type:**

```
- `bool`: Represents logical values: `True` or `False`. Used for conditional
statements and boolean expressions.
```

- Example #1: Exploring Boolean Values and Type Checking with isinstance() and bool() functions

```python
# Boolean (bool) for a true/false condition
is_raining = True

# Using booleans in an if statement
if is_raining:
    print("Bring an umbrella!")
```

Great! Here's your next task.

Task 1: Type Checking in Python

Write a Python program that:

- Declares three different types of numbers: an integer, a float, and a complex number.
- Uses the `type()` function to check the data type of each number.
- Prints the data type of each number in a descriptive way.

Example Output:

```
The type of variable x is: <class 'int'>
The type of variable y is: <class 'float'>
The type of variable z is: <class 'complex'>
```

Task 2: Convert Integer to String

Write a Python program that:

- Takes an integer as input.
- Converts it to a string using the `str()` function.
- Concatenates the string with another message and prints it.

Example Input:

```
Enter a number: 100
```

Expected Output:

```
Your number is: 100
```

Task 3: Convert String to Integer

Write a Python program that:

- Takes a numeric string as input (e.g., `"123"`).
- Converts it into an integer using the `int()` function.
- Multiplies the number by 2 and prints the result.

Example Input:

```
Enter a number as a string: 50
```

Expected Output:

```
Double the number: 100
```

Task 4: Convert Integer to Octal and Hexadecimal

Write a Python program that:

- Takes an integer as input from the user.
- Converts it to **octal** using `oct()` and **hexadecimal** using `hex()`.
- Prints both results.

Example Input:

```
Enter a number: 255
```

Expected Output:

```
Octal: 0o377
Hexadecimal: 0xff
```

Task 5: Convert Octal and Hexadecimal to Integer

Write a Python program that:

- Takes an **octal** and a **hexadecimal** number as input (as strings).

- Converts them back to integers using `int(value, base)`.
- Prints the decimal values.

Example Input:

```
Enter an octal number: 0o377
Enter a hexadecimal number: 0xff
```

Expected Output:

```
Octal to Integer: 255
Hexadecimal to Integer: 255
```

3.4 **Data Structures and Sequences**

3.4.1 Tuple in Python

- In python, a tuple is an immutable sequence of elements. it is similar to a list, but the elements of a tuple cannot be modified once they are created.
- Tuple is a collection data type in python. It is useful for storing multiple related values as a single unit.
- Sequence types in python - list, tuple and range

Creating a Tuple in Python

A tuple is created by enclosing elements within parentheses () and separating them with commas.
While parentheses are technically optional, it's generally considered best practice to use them for clarity and consistency.

- video: How to create a tuple in Python

Example

Some common ways to create tuples in Python include:

```
tup = (1,2,3)
print(tup) # Output: (1, 2, 3)
# check the type of variable
print(type(tup)) # Output: <class 'tuple'>

# another example to create tuple
tup1 = 4,5,6
print(tup1) # Output: (4, 5, 6)

# tuple with mixed datatypes
tup_mixed = (7, "String", 7.8)
print(tup_mixed)

tup4 = tuple('string')
```

```
print(tup4) # Output: ('s','t','r','i','n','g')
```

3.4.2 List

- A **list** in Python is one of the most commonly used data structures. It allows you to store a collection of items (which can be of different types) in a single variable. Lists are very flexible and easy to use, making them a great tool for beginners to understand.

Creating a List

You can create a list by placing items inside square brackets `[]`, separated by commas.

```
# A list of integers
numbers = [1, 2, 3, 4, 5]

# A list of strings
fruits = ["apple", "banana", "cherry"]

# A list of mixed data types
mixed_list = [1, "hello", 3.14, True]

# An empty list
empty_list = []
```

Accessing Elements in a List

You can access individual elements in a list using their index.

```
# Access the first element (index 0)
print(fruits[0])  # Output: apple

# Access the second element (index 1)
print(fruits[1])  # Output: banana

# Access the last element (index -1)
print(fruits[-1])  # Output: cherry
```

Modifying Elements in a List

Since lists are mutable, you can change an element in a list by assigning a new value to a specific index.

```
# Change the first element of the list
fruits[0] = "orange"
print(fruits)  # Output: ['orange', 'banana', 'cherry']
```

3.4.3 Introduction to Python Dictionaries: Concepts, Usage, and Examples

In Python, a **dictionary** is a collection of key-value pairs. Each key in a dictionary is unique, and it is associated with a value. Dictionaries are used to store data values like a map or a real-life dictionary where each word (key) has a definition (value). They are mutable, meaning you can change, add, or remove items after the dictionary is created.

Creating a Dictionary

You create a dictionary using curly braces `{}` with keys and values separated by a colon `:`. Multiple key-value pairs are separated by commas.

```
# Example of a dictionary
student = {
    "name": "John",
    "age": 20,
    "courses": ["Math", "Science"]
}
```

Accessing Values

You can access the value associated with a specific key by using square brackets `[]` or the `get()` method.

```
# Accessing values
print(student["name"])  # Output: John
print(student.get("age"))  # Output: 20
```

Adding or Updating Elements

You can add a new key-value pair or update an existing one by assigning a value to the key.

```
# Adding a new key-value pair
student["grade"] = "A"

# Updating an existing value
student["age"] = 21
```

3.4.4 Sets

In Python, a **set** is an unordered collection of unique elements, meaning no duplicates are allowed. Sets are useful when you want to store multiple items but don't need to keep them in a particular order, and you want to ensure that each item only appears once.

Creating a Set

You can create a set using curly braces `{}` or the `set()` function.

```
# Creating a set using curly braces
my_set = {1, 2, 3, 4, 5}

# Creating a set using the set() function
my_set = set([1, 2, 3, 4, 5])
```

Example:

```
# Creating a set
fruits = {"apple", "banana", "cherry", "apple"}

# Displaying the set
print(fruits)  # Output: {'apple', 'banana', 'cherry'}
```

Notice how `apple` only appears once, even though we tried to add it twice.

For more details , see Data Structures and Sequences

Task 6: Creating a Tuple

Write a Python program that:

- Creates a tuple with **three** different items (e.g., a number, a word, and a decimal).
- Prints the **entire tuple**.
- Prints the **first item** in the tuple.

Example Output:

```
Tuple: (10, 'Apple', 3.5)
First item: 10
```

Task 7: Tuple Unpacking

Write a Python program that:

- Creates a tuple with three values: name, age, and country.
- Uses **tuple unpacking** to assign each value to separate variables.
- Prints the extracted values.

Example Output:

```
Name: Alice
Age: 25
Country: USA
```

Task 8: Accessing Tuple Elements

Write a Python program that:

- Creates a tuple with **five numbers**.
- Prints the **last number** using negative indexing.

Example Output:

```
Numbers: (5, 10, 15, 20, 25)
Last number: 25
```

Task 9: Creating a List

Write a Python program that:

- Creates a list of **four favorite foods**.
- Prints the **entire list**.
- Prints the **second food** from the list.

Example Output:

```
Favorite Foods: ['Pizza', 'Burger', 'Pasta', 'Ice Cream']
Second food: Burger
```

Task 10: Changing an Item in a List

Write a Python program that:

- Creates a list with **three colors**.
- Changes the **first color** to a new one.
- Prints the updated list.

Example Output:

```
Original List: ['Red', 'Blue', 'Green']
Updated List: ['Yellow', 'Blue', 'Green']
```

Task 11: Adding to a List

Write a Python program that:

- Creates an empty list.

- Asks the user to enter **three city names** and adds them to the list.
- Prints the final list.

Example Input:

```
Enter a city: Paris
Enter a city: London
Enter a city: Tokyo
```

Example Output:

```
Cities: ['Paris', 'London', 'Tokyo']
```

Task 12: Creating a Dictionary

Write a Python program that:

- Creates a dictionary with **two items** (name and age).
- Prints the dictionary.
- Prints only the **name** from the dictionary.

Example Output:

```
Person: {'name': 'Alice', 'age': 25}
Name: Alice
```

Task 13: Adding to a Dictionary

Write a Python program that:

- Creates a dictionary with **name and country**.
- Adds a new key **"hobby"** with a value.
- Prints the updated dictionary.

Example Output:

```
Original Dictionary: {'name': 'John', 'country': 'USA'}
Updated Dictionary: {'name': 'John', 'country': 'USA', 'hobby': 'Reading'}
```

Task 14: Creating a Set

Write a Python program that:

- Creates a set with **four different numbers**.
- Prints the set.

Example Output:

```
Numbers: {1, 2, 3, 4}
```

Task 15: Removing Duplicates Using a Set

Write a Python program that:

- Takes a list with **duplicate numbers**.
- Converts it into a set to remove duplicates.
- Prints the unique numbers.

Example Input:

```
Original List: [1, 2, 2, 3, 3, 4, 5]
```

Example Output:

```
Unique Numbers: {1, 2, 3, 4, 5}
```

Task 16: Creating and Accessing a List

Write a Python program that:

- Creates a list of **five favorite movies**.
- Prints the **entire list**.
- Accesses and prints the **third movie** in the list using indexing.
- Changes the **last movie** in the list to a new movie.
- Prints the updated list.

Example Output:

```
Original List: ['Inception', 'Titanic', 'Avatar', 'Interstellar', 'Joker']
Third movie: Avatar
Updated List: ['Inception', 'Titanic', 'Avatar', 'Interstellar', 'The Dark
Knight']
```

Task 7: Creating and Using a Set

Write a Python program that:

- Creates a set of **unique numbers** from a given list (including duplicate values).
- Prints the unique numbers.

Example Input:

```
Original List: [1, 2, 2, 3, 4, 4, 5]
```

Expected Output:

```
Unique Numbers: {1, 2, 3, 4, 5}
```

Task 8: Set Operations

Write a Python program that:

- Creates two sets: one with **even numbers** and one with **prime numbers** (both from 1 to 10).
- Finds the **union** (all unique numbers from both sets).
- Finds the **intersection** (numbers that are in both sets).
- Prints the results.

Expected Output:

```
Even Numbers: {2, 4, 6, 8, 10}
Prime Numbers: {2, 3, 5, 7}
Union: {2, 3, 4, 5, 6, 7, 8, 10}
Intersection: {2}
```

3.5 Why Use Data Types?

video: 3 Reasons Why Are Data Types So Important in Python

Data types are essential in Python for several reasons:

- **Memory Management:** Different data types use memory in different ways. Knowing the type helps Python allocate the right amount of memory. For example, an integer requires less space than a string or a list.
  - video: How to Get the Size of an Object in Bytes | Python Tutorial for Beginners
- **Type Safety:** Data types help prevent errors by ensuring operations are compatible with the data being used. You can't add a string to an integer, for instance.

- **Readability:** Using appropriate data types makes code easier to understand. It's clear what kind of data a variable holds and how it can be used.
- **Performance:** Python can optimize certain operations based on the data type. For example, mathematical calculations on integers are faster than on floats.

3.6 Understanding Dynamic Variables in Python with Examples

video: Is Python a Dynamic Language?

- Python is a dynamically typed language. This means that the Python interpreter does type checking only as code runs, and the type of a variable is allowed to change over its lifetime.[1]
- In Python, variables are dynamic, meaning they can change types during the execution of a program. This flexibility allows you to assign a value of any type to a variable and later reassign it to a value of a different type without any issues. This dynamic nature of variables is due to Python being a dynamically typed language.

Example #: Dynamic Variables in Python

```
# Initial assignment of an integer value
x = 10
print(x)  # Output: 10
print(type(x))  # Output: <class 'int'>

# Reassigning a string value to the same variable
x = "Hello, World!"
print(x)  # Output: Hello, World!
print(type(x))  # Output: <class 'str'>

# Reassigning a list to the same variable
x = [1, 2, 3]
print(x)  # Output: [1, 2, 3]
print(type(x))  # Output: <class 'list'>

# Reassigning a float value to the same variable
x = 3.14
print(x)  # Output: 3.14
print(type(x))  # Output: <class 'float'>
```

In this example:

1. `x` is initially assigned an integer value of `10`.
2. `x` is then reassigned a string value `"Hello, World!"`.
3. `x` is later reassigned a list `[1, 2, 3]`.
4. Finally, `x` is reassigned a float value `3.14`.

Each time, the type of `x` changes dynamically to match the type of the value assigned to it. This flexibility is one of the powerful features of Python, allowing for more concise and adaptable code.

3.7 **Python `input()` Function - Lecture Notes**

What is `input()`?

The `input()` function in Python is used to take user input from the keyboard. It allows a program to interact with users by asking for information.

Syntax:

```
variable_name = input("Prompt message")
```

- `Prompt message`: A string displayed to the user before they enter input.
- `variable_name`: The variable where the user's input is stored.

Basic Example

```
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

Explanation:

1. `input("Enter your name: ")` displays the message **"Enter your name: "** and waits for the user to type something.
2. The user types their name and presses Enter.
3. The input is stored in the variable `name`.
4. `print("Hello, " + name + "!")` displays a greeting message with the user's name.

Taking Numerical Input

By default, `input()` returns a string. If you need a number, you must convert it using `int()` or `float()`.

Example:

```
age = int(input("Enter your age: "))
print("In 5 years, you will be", age + 5)
```

Explanation:

1. The user enters their age as a string.
2. `int(input(...))` converts it into an integer.
3. The program calculates the age after 5 years and prints it.

None

In Python, `None` is a special constant that represents the absence of a value or a null value. It is an object of its own datatype, called `NoneType`.

Examples:

1. **Assigning `None` to Variables**:

```
a = None
```

2. **Checking for `None`**:

```
if a is None:
    print("a is None")
else:
    print("a is not None")
```

- Python Quiz -String
- Python Quiz - Scalar Types

Type Hints

In Python, **type hints** allow you to specify the expected data types of variables, function parameters, and return values. They make the code more readable and help developers understand what kind of values are expected.

Here's how you can use type hints in Python:

1. **Type Hinting for Variables**

You can add type hints to variables by using a colon `:` after the variable name, followed by the type:

Example: Type Hinting for Variables

```
age: int = 25
name: str = "Alice"
height: float = 5.7
is_student: bool = True
```

2. Type Hinting in Functions

For functions, type hints are added after the parameter names and before the return type with `->`.

Example: Type Hinting in Functions

```
def greet(name: str) -> str:
    return f"Hello, {name}!"

# Usage
print(greet("Alice"))  # Output: Hello, Alice!
```

This code specifies that the `name` parameter should be a `str`, and the function should return a `str`.

3. **Type Hints for Collections**

For more complex types like lists, dictionaries, sets, and tuples.

Lists

To specify that a list contains elements of a certain type, use `list`.

```
# A list of integers
numbers: list[int] = [1, 2, 3, 4, 5]

# A list of strings
names: list[str] = ["Alice", "Bob", "Charlie"]
```

Dictionaries

For dictionaries, you can specify the types of both keys and values using `dict`.

Example:

```
# A dictionary with string keys and integer values
age_map: dict[str, int] = {"Alice": 30, "Bob": 25, "Charlie": 35}
```

Sets

To specify the type of elements in a set, use `Set`.

Example:

```
# A set of strings
unique_names: set[str] = {"Alice", "Bob", "Charlie"}
```

Functions with Collection Type Hints

You can also use type hints in function definitions to specify the types of parameters and return values.

Example:

```
# Function that processes a list of integers and returns a dictionary
def process_data(numbers: list[int]) -> dict[str, int]:
    result = {
        'sum': sum(numbers),
        'count': len(numbers)
    }
    return result

data = [1, 2, 3, 4, 5]
processed_data = process_data(data)
print(processed_data)  # Output: {'sum': 15, 'count': 5}
```

Using type hints doesn't enforce types at runtime but can improve code readability and help detect type-related issues with tools like **mypy**.

Key Terms

Fix the error

Incorrect Input Conversion [Python Quiz #64]

```
age = input("Enter your age: ")
result = age * 2
```

True/False (Mark T for True and F for False)

1. In Python, the type of a variable is determined at runtime.

Answer Key (True/False):

1. True

Multiple Choice (Select the best answer)

What is the output of the following code? [Python Quiz #22]

```
age = 25
print("I am " + str(age) + " years old.")
```

- A) I am 25 years old.
- B) I am "25" years old.
- C) Syntax error

- D) I am years old. (with quotes)

Watch this video for the answer:https://youtube.com/shorts/DBC5-ZYoGXI?si=PXk-CPGymx2Q6X2p

```
age = 25
message = "You are " + str(age) + " years old."
message += " Welcome!"
print(message)
```

1. **Which function would you use to determine the type of a variable in Python?**
    - A) id()
    - B) type()
    - C) str()
    - D) isinstance()
2. **Which of the following is not a scalar data type in Python?**
    - A) bool
    - B) int
    - C) float
    - D) list
3. Which data type is used to represent decimal numbers in Python?
    - A) int
    - B) float
    - C) complex
    - D) str
4. Which of the following is an example of a boolean value in Python?
    - a. "True"
    - b. 1
    - c. 3.14
    - d. False
5. Which scalar data type is used to represent textual data in Python?

- a. str
- b. char
- c. text
- d. string

1. What is the default type of a numerical literal without a decimal point in Python?

- a. int
- b. float
- c. complex
- d. bool

1. **What is the result of the expression type("Hello, World!") in Python?**
    - A) <class 'str'>
    - B) <class 'bool'>
    - C) <class 'int'>
    - D) <class 'float'>

2. What is the output of type(42)?
    - A) <class 'str'>
    - B) <class 'bool'>
    - C) <class 'int'>
    - D) <class 'float'>

3. What is the result of 3 + 4.5?

- a. 7
- b. 7.5
- c. Error
- d. None of the above

1. How do you create a string in Python?
    - A) Using single quotes (')
    - B) Using double quotes (")
    - C) Both a and b
    - D) None of the above

2. Which of the following is a valid boolean value in Python?

- a. True
- b. False
- c. 0
- d. All of the above

1. What is the output of str(3.14)?

- a. 3.14
- b. '3.14'
- c. Error
- d. None of the above

1. What is the result of the following expression? [Python Quiz #79]

```
type(3 + 4.0)
```

```
- A) <class 'str'>
- B) <class 'bool'>
```

```
- C) <class 'int'>
- D) <class 'float'>
```

14. Which of the following is a correct way to declare a complex number in Python?

    - A) a = 3 + 4j
    - B) a = 3.4j
    - C) a = 3 + 4i
    - D) a = 3 + 4

15. Which function can be used to convert a float to an integer in Python?

    - A) float()
    - B) int()
    - C) str()
    - D) bool()

16. Which of the following statements is true regarding dynamic typing in Python?

    - A) Variables can only be assigned values of the same type.
    - B) The data type of a variable is determined at runtime based on the value it holds.
    - C) Variables must be declared with a specific type.
    - D) Once a variable is assigned a type, it cannot be changed.

17. In Python, what happens if you assign a new value of a different type to a variable?

    - A) Python will raise a type error.
    - B) Python will change the variable's type to match the new value.
    - C) Python will ignore the new value.
    - D) Python will convert the value to the original type.

18. What is the output of the following code? [Python Quiz #80]

```
x = 10
x = "Hello"
print(type(x))
```

```
- A) <class 'int'>
- B) <class 'str'>
- C) <class 'bool'>
- D) <class 'float'>
```

19. What is the result of the following code?

```
x = 5
x = 5.0
```

```
x = True
x = "Python"
print(x)
```

```
- A) 5
- B) 5.0
- C) True
- D) Python
```

20. What is the main advantage of dynamic typing in Python?

    - A) Faster execution time.
    - B) More flexibility in code.
    - C) Improved error detection at compile-time.
    - D) Reduced memory usage.

21. Which of the following best describes a dynamically typed language?

    - A) Type checking is performed during code compilation.
    - B) Type checking is deferred until program execution.
    - C) Type checking is not performed at all.
    - D) Types are always explicitly declared by the programmer.

22. In Python, which of the following is true about variable assignment?

    - A) The type of the variable is determined when the variable is first assigned a value.
    - B) The type of the variable is determined at compile-time.
    - C) Variables must be explicitly typed before assignment.
    - D) Variables cannot change type once assigned.

23. What is the output of the following code? [Python Quiz #81]

```
x = "10"
x = int(x) + 2
print(x)
```

```
- A) "102"
- B) 102
- C) 12
- D) "12"
```

Fill in the Blanks

Answer Key (Fill in the Blanks):

Exercises

Exercise 1: Variable Assignment and Basic Operations

1. Assign the value `5` to a variable named `x`.
2. Assign the value `10` to a variable named `y`.
3. Assign the sum of `x` and `y` to a variable named `sum_xy`.
4. Print the value of `sum_xy`.

Solution:

```
x = 5
y = 10
sum_xy = x + y
print("Sum of x and y:", sum_xy)
```

Exercise 2: Working with Different Data Types

1. Assign a floating-point number to a variable named `pi`.
2. Assign a string to a variable named `greeting`.
3. Assign a boolean value to a variable named `is_active`.
4. Print the types and values of `pi`, `greeting`, and `is_active`.

Solution:

```
pi = 3.14
greeting = "Hello, World!"
is_active = True

print("Type of pi:", type(pi), "Value:", pi)
print("Type of greeting:", type(greeting), "Value:", greeting)
print("Type of is_active:", type(is_active), "Value:", is_active)
```

Exercise 3: String Concatenation

1. Assign your first name to a variable named `first_name`.
2. Assign your last name to a variable named `last_name`.
3. Concatenate `first_name` and `last_name` with a space in between and assign the result to a variable named `full_name`.
4. Print the value of `full_name`.

Solution:

```
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print("Full name:", full_name)
```

Exercise 4: Boolean Operations

1. Assign the value `True` to a variable named `is_sunny`.
2. Assign the value `False` to a variable named `is_raining`.
3. Create a new variable named `can_go_outside` that is `True` if `is_sunny` is `True` and `is_raining` is `False`.
4. Print the value of `can_go_outside`.

Solution:

```
is_sunny = True
is_raining = False
can_go_outside = is_sunny and not is_raining
print("Can go outside:", can_go_outside)
```

Exercise 5: Type Conversion

1. Assign the string `"123"` to a variable named `num_str`.
2. Convert `num_str` to an integer and assign it to a variable named `num_int`.
3. Print the type and value of `num_int`.

Solution:

```
num_str = "123"
num_int = int(num_str)
print("Type of num_int:", type(num_int), "Value:", num_int)
```

Review Questions

Basic Data Types

1. What are the basic data types in Python?
    - **Answer:** Python's basic data types include:
        1. **Numbers:** Integers (`int`), floating-point numbers (`float`), and complex numbers (`complex`).
        2. **Text:** Strings (`str`) to represent sequences of characters.
        3. **Logical Values:** Booleans (`bool`) for True or False.
2. How do you create a string in Python? Give an example.
3. What is the difference between an integer and a float in Python?
4. How do you convert a string to an integer in Python?
5. How can you convert a string to an integer and an integer to a string in Python? Provide examples.

6. What function would you use to find the type of a variable in Python?

7. What is the purpose of the type() function in Python?

Boolean and None

6. What are the Boolean values in Python?

7. What does the None type represent in Python?

   - **Answer:** In Python, the None type represents the absence of a value or a null value. It is a built-in constant that is used to denote a lack of value or a null reference.

   - Characteristics of `None`

     1. **Singleton**: `None` is a singleton in Python, meaning there is only one instance of `None` in a Python runtime. All occurrences of `None` point to the same object.

```
a = None
b = None
print(a is b)  # Output: True
```

     2. **Type**: The type of `None` is `NoneType`.

```
print(type(None))  # Output: <class 'NoneType'>
```

     3. **Boolean Context**: `None` is treated as `False` in a boolean context.

```
if not None:
    print("None is considered False")  # Output: None is considered False
```

   - Checking for `None`

   - To check if a variable is `None`, use the `is` operator, as it checks for identity.

```
variable = None
if variable is None:
    print("Variable is None")  # Output: Variable is None
```

8. How do you check if a variable is None?

   - **Answer:** To check if a variable is None in Python, you should use the is operator. The is operator checks for identity, meaning it checks whether two references point to the same object. Since

None is a singleton in Python (there is only one instance of None in a Python runtime), using is is the correct and most efficient way to check for None.

```
variable = None

if variable is None:
    print("Variable is None")
```

Type Casting

1. How do you convert an integer to a string in Python?
2. What is the result of int('123.45')?
3. What does the str() function do?
4. How do you safely convert a string to a float, considering the possibility of invalid input?

References and Bibliography

[1]R. Python, “Dynamic vs Static – Real Python,” realpython.com. https://realpython.com/lessons/dynamic-vs-static/ [2]Python Software Foundation, “Built-in Types — Python 3.12.1 documentation,” Python.org, 2019. https://docs.python.org/3/library/stdtypes.html [3]“PEP 526 – Syntax for Variable Annotations | peps.python.org,” peps.python.org. https://peps.python.org/pep-0526/