Python: Data Types

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

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

Here are some of the fundamental built-in data types in Python:

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
```

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. **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

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

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

Why Use Data Types?

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.

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.

- Python Quiz -String
- Python Quiz - Scalar Types

Key Terms

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)

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

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

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

7. **What is the result of the expression type("Hello, World!") in Python?**

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

8. What is the output of type(42)?

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

9. What is the result of 3 + 4.5?

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

10. 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

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

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

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

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

13. What is the result of the following expression?

```
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?

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

```
- A) <class 'int'>
- B) <class 'str'>
- C) <class 'bool'>
- D) <class 'floa   - t'>
```

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?

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