ICT @ Mzuni

Programming with Python

by: Chimango Nyasulu, PhD

ICT @ Mzuni

Outline

- Variables
- Basic Data Types (int, float, double, String)
- Immutable Variables
- Constants
- Arithmetic Operators and Expressions
- Documenting the code

Variables in Python

- Containers for storing data
- Created when you assign a value
- No need to declare data type explicitly

Example:

x = 10
name = "Alice"

ICT @ Mzuni

Basic Data Types in Python

- int: Whole numbers
- float: Decimal numbers
- str: Strings or text
- Python does not have a separate double type; float is double precision

Immutable Variables

- Immutable variables are objects whose values cannot be changed after they are created.
- Once an immutable object is created, any modification to it results in the creation of a new object with the updated value, rather than altering the original object.

Examples of Immutable Types in Python

- Integers, Strings, Tuples, Booleans, Frozen Sets

Why Use Immutable Variables?

- Immutable objects ensure that data remains consistent and unaltered.
- They are safe to use in multi-threaded environments.
- Can be used as dictionary keys or set elements because their hash value does not change.

ICT @ Mzuni

Immutable Variables

```
1  s = "Hello"
2  print(id(s))  # Memory address of s
3  s += " World"  # A new string object is created
4  print(id(s))  # Memory address changes
```

```
1  x = 10
2  print(id(x))  # Memory address of x
3  x += 5        # A new integer object is created
4  print(id(x))  # Memory address changes
```

ICT @ Mzuni

Constants in Python

- Constants are variables whose values are not intended to change during the execution of a program.
- Python does not have built-in support for constants (unlike some other languages like C++ or Java)
- Developers use naming conventions and immutability principles to define constants.

Examples of Constants in Python

```
1  # constants.py
2  PI = 3.14159
3  MAX_USERS = 100
```

Arithmetic Operators and Expressions

- Python provides several arithmetic operators that allow you to perform mathematical operations on numerical values.

- These operators are essential for performing calculations and creating expressions in Python.

Basic arithmetic operators are:

- + (Addition)
- - (Subtraction)
- * (Multiplication)
- / (Division)
- // (Floor Division)
- % (Modulus - Remainder)
- ** (Exponentiation)

ICT @ Mzuni

Arithmetic Operators and Expressions

```python
# Addition
result = 5 + 3
print(result)  # Output: 8


# Subtraction
result = 10 - 4
print(result)  # Output: 6
```

```python
# Mixed operations
result = (10 + 5) * 2  # Parentheses take precedence
print(result)  # Output: 30


# Division and modulus
result = 10 / 3  # Returns a float
remainder = 10 % 3  # Remainder of division
print(result, remainder)  # Output: 3.333..., 1
```

ICT @ Mzuni

Arithmetic Operators and Expressions

```
# Powers of numbers
result = 2 ** 3  # 2 raised to the power of 3
print(result)  # Output: 8
```

```
# Integer division
result = 9 // 2  # Divides and rounds down to the nearest integer
print(result)  # Output: 4
```

Arithmetic Operators and Expressions

- Python follows a specific order of operations, known as operator precedence, to evaluate expressions.

- The precedence rules are similar to those in mathematics:

1. Parentheses (): Highest precedence.
2. Exponentiation **.
3. Multiplication *, Division /, Floor Division //, Modulus %.
4. Addition +, Subtraction -.

```
result = 10 + 5 * 2  # Multiplication is evaluated first
print(result)  # Output: 20


result = (10 + 5) * 2  # Parentheses take precedence
print(result)  # Output: 30
```

Documenting the Code

- Documentation is a critical aspect of writing clean, maintainable, and reusable code.
- In Python, documenting your code involves adding comments, docstrings, and other annotations to make it easier for others (and your future self) to understand the purpose and functionality of your code.

Why Document Code?

- Helps explain the logic and purpose of the code.
- Makes it easier for others to contribute to or use your code.
- Simplifies debugging, updating, and extending the code.
- Well-documented code reflects good coding practices.