Variables

In Python, variables are used to store data in the computer's memory. Each variable has a name that allows you to reference and access the data it holds. Unlike some other programming languages, Python is dynamically typed, which means you don't need to explicitly declare the data type of a variable. Instead, Python infers the data type from the assigned value. Here's a detailed explanation of Python variables with examples:

1. Variable Declaration and Assignment: In Python, you can create a variable by giving it a name and assigning a value to it using the assignment operator `=`. For example:

```
# Variable declaration and assignment
x = 10
```

In this example, we created a variable named `x` and assigned the integer value `10` to it. Python automatically determines that `x` is an integer because of the assigned value.

2. Dynamic Typing: As mentioned earlier, Python is dynamically typed, which means you can change the data type of a variable by assigning a different value to it. For example:

```
# Dynamic typing
x = 10  # x is an integer
x = "hello"  # Now x is a string
```

3. Variable Names: Variable names in Python can consist of letters (both uppercase and lowercase), digits, and underscores. They must start with a letter or an underscore, but they cannot start with a digit. Variable names are case-sensitive, so `x`, `X`, and `_x` are considered different variables.

```
# Valid variable names
my_variable = 42
user_name = "John"
_temperature = 25.5
```

```
# Invalid variable names
2nd_var = 100  # Cannot start with a digit
user-name = "Jane"  # Hyphen is not allowed in variable names
```

4. Data Types and Type Inference: As mentioned earlier, Python infers the data type based on the assigned value. Some common data types in Python include integers, floats, strings, booleans, lists, tuples, sets, and dictionaries.

```
# Type inference
x = 10  # integer
y = 3.14  # float
name = "Alice"  # string
is_student = True  # boolean
my_list = [1, 2, 3]  # list
my_tuple = (10, 20, 30)  # tuple
my_set = {1, 2, 3}  # set
my_dict = {'name': 'Bob', 'age': 30}  # dictionary
```

5. Variable Scope: Variables in Python have a scope, which defines where the variable is accessible. Variables defined outside any function have global scope, whereas variables defined inside a function have local scope.

```
# Global scope
global_var = 100

def my_function():
    # Local scope
    local_var = 42
    print("Inside the function:", local_var)

print("Outside the function:", global_var)
my_function()
```

6. Variable Reassignment: You can reassign a new value to an existing variable at any time.

```
# Variable reassignment
x = 10
print(x)  # Output: 10

x = 20
print(x)  # Output: 20
```

Variables are a fundamental concept in programming and play a crucial role in storing and manipulating data throughout your Python programs. With dynamic typing, Python provides flexibility in handling variables and allows you to work with various data types without the need for explicit type declarations.

Types Of Variable

In Python, there are several types of variables based on their scope and where they are defined. The most common types of variables in Python are:

1. Local Variables:
    - Local variables are defined within a function and have a local scope.
    - They are accessible only inside the function where they are defined.
    - Local variables are created when the function is called and destroyed when the function exits.
    - They cannot be accessed from outside the function.

Example of a local variable:

```
def my_function():
    x = 10  # This is a local variable
    print(x)

my_function()  # Output: 10
# print(x)  # This will raise an error since x is not defined in the global scope
```

2. Global Variables:
    - Global variables are defined outside any function and have a global scope.
    - They are accessible from anywhere in the program, both inside and outside functions.
    - Global variables are created when the program starts and destroyed when the program ends.
    - They can be modified from within functions, but you need to use the `global` keyword to explicitly indicate that you want to modify the global variable.

Example of a global variable:

```
global_var = 100  # This is a global variable

def my_function():
    print(global_var)  # Accessing the global variable

my_function()  # Output: 100

def modify_global():
    global global_var  # Use the 'global' keyword to modify the global variable
    global_var = 200

modify_global()
print(global_var)  # Output: 200
```

3. Class Variables (Static Variables):

- Class variables are defined inside a class but outside any method. They belong to the class and are shared among all instances (objects) of the class.
- Class variables have the same value for all instances of the class.
- You can access class variables using the class name or instance variables using the `self` keyword inside class methods.

Example of a class variable:

```
class MyClass:
    class_var = "I am a class variable"

    def __init__(self):
        self.instance_var = "I am an instance variable"

obj1 = MyClass()
obj2 = MyClass()

print(obj1.class_var)  # Output: "I am a class variable"
print(obj2.class_var)  # Output: "I am a class variable"

obj1.class_var = "Modified class variable for obj1"

print(obj1.class_var)  # Output: "Modified class variable for obj1"
print(obj2.class_var)  # Output: "I am a class variable"
```

4. Instance Variables:

- Instance variables are defined within the constructor method (`__init__`) of a class.
- Each instance (object) of the class has its own copy of instance variables.
- Instance variables represent the attributes of each object and can have different values for each object.

Example of an instance variable:

```
class Student:
    def __init__(self, name, age):
        self.name = name  # This is an instance variable
        self.age = age

student1 = Student("Alice", 20)
student2 = Student("Bob", 22)

print(student1.name)  # Output: "Alice"
print(student2.name)  # Output: "Bob"
```

These are the main types of variables in Python based on their scope and where they are defined. Understanding the types of variables is important for proper variable management and avoiding scope-related issues in your Python programs.

Keywords Related To Variables

Here are some keywords and concepts related to variables in Python:

1. `global`: This keyword is used to declare a variable in the global scope from within a function. It allows you to modify a global variable from inside a function.
2. `nonlocal`: This keyword is used to declare a variable in the nearest enclosing scope that is not global. It allows you to modify a variable in the outer (enclosing) scope from within a nested function.
3. `self`: In the context of classes, `self` is used as the first parameter of instance methods to refer to the instance (object) itself. It is used to access and modify instance variables within class methods.
4. `class`: This keyword is used to define a class, which serves as a blueprint for creating objects. Class variables and instance variables are defined within a class.
5. `def`: This keyword is used to define a function. Functions can contain local variables that are accessible only within the function's scope.
6. `lambda`: This keyword is used to create anonymous (nameless) functions using the `lambda` syntax. Lambdas can capture variables from their surrounding scope.
7. `del`: This keyword is used to delete variables or items from data structures like lists or dictionaries. When you use `del` with a variable name, it removes the reference to the variable, and the memory occupied by the variable is released.

Here's an example illustrating the use of some of these keywords:

```
global_var = 100  # Global variable

def my_function():
    nonlocal_var = 50  # Enclosing scope variable

    def inner_function():
        nonlocal nonlocal_var  # Use 'nonlocal' to modify the enclosing scope variable
        nonlocal_var = 60

    inner_function()
    print("Enclosing scope variable:", nonlocal_var)  # Output: 60

class MyClass:
    class_var = "I am a class variable"  # Class variable
    def __init__(self, name):
        self.instance_var = name  # Instance variable

    def display(self):
        print("Instance variable:", self.instance_var)

obj = MyClass("Alice")
obj.display()  # Output: "Instance variable: Alice"

del global_var  # Delete the global variable
# print(global_var)  # This will raise an error since the variable is deleted

del obj.instance_var  # Delete the instance variable
# obj.display()  # This will raise an error since the instance variable is deleted
```

Summary

Topic: Variable Basics

1. Variable Declaration and Assignment
    - Variables are created by assigning a value to a name using the `=` operator.
    - No explicit type declaration is required in Python; the data type is inferred from the assigned value.

Topic: Variable Naming

1. Variable Names
    - Variable names can contain letters (both uppercase and lowercase), digits, and underscores.
    - They must start with a letter or an underscore, but they cannot start with a digit.
    - Variable names are case-sensitive.

Topic: Variable Data Types and Type Inference

1. Data Types and Type Inference
    - Python supports various data types, including integers, floats, strings, booleans, lists, tuples, sets, and dictionaries.
    - The data type of a variable is inferred from the assigned value.

Topic: Variable Scope

1. Local Scope
    - Variables defined inside a function are accessible only within that function.

   - Local variables are created when the function is called and destroyed when it exits.

2. Global Scope
   - Variables defined outside any function are accessible throughout the entire program.
   - Global variables are created when the program starts and destroyed when the program ends.

3. Class Scope
   - Variables defined within a class but outside any method are shared among all instances of the class.
   - Class variables are created when the class is defined and destroyed when the program ends.

Topic: Dynamic Typing

1. Dynamic Typing
   - Python is dynamically typed, meaning the data type of a variable can change during runtime.
   - Variables can be reassigned with new values of different data types.

Topic: Global and Nonlocal Keywords

1. Global Keyword
   - Used to declare a variable in the global scope from within a function.
   - Allows modifying a global variable from inside a function.

2. Nonlocal Keyword
   - Used to declare a variable in the nearest enclosing scope that is not global.
   - Allows modifying a variable in the outer (enclosing) scope from within a nested function.

Topic: Instance Variables and Class Variables

1. Instance Variables
   - Represent the attributes of each object and have different values for each instance.
   - Defined within the constructor method (`__init__`) of a class using the `self` keyword.

2. Class Variables (Static Variables)
   - Belong to the class and are shared among all instances (objects) of the class.
   - Defined within a class but outside any method.

Topic: Deleting Variables

1. Deleting Variables

- The `del` keyword is used to delete variables, freeing up the memory occupied by the variable.

Topic: Mutable and Immutable Variables

1. Mutable Variables
    - Data types like lists and dictionaries are mutable, allowing their values to be changed after creation.
2. Immutable Variables
    - Data types like integers, floats, strings, and tuples are immutable and cannot be changed once created.

Topic: Variable Conventions

1. Variable Naming Conventions
    - Python follows certain naming conventions for variables (PEP 8).
    - Variable names are typically written in lowercase, and multiple words are separated by underscores (snake_case).