Python_Basic

Thursday, November 7, 2024

Example of Different Types in Action

Here's a sample Python program demonstrating these different types:

```
# Integer
age = 30

# Float
temperature = 98.6

# String
name = "Alice"

# Boolean
is_member = True

# List
fruits = ["apple", "banana", "cherry"]

# Tuple
coordinates = (10, 20)

# Dictionary
person = {
    "name": "Alice",
    "age": 25,
    "city": "New York"
}

# Set
unique_numbers = {1, 2, 3, 4, 5}

# NoneType
result = None

print("Integer:", age)
print("Float:", temperature)
print("String:", name)
print("Boolean:", is_member)
print("List:", fruits)
print("Tuple:", coordinates)
print("Dictionary:", person)
print("Set:", unique_numbers)
print("NoneType:", result)
```

1. Integer (`int`)

- Represents whole numbers (positive, negative, or zero).
- Examples: `5`, `-10`, `0`
- **Example in Python:**

```
age = 25
count = -10
year = 2023
```

2. Floating Point (`float`)

- Represents decimal numbers.
- Examples: `3.14`, `-0.001`, `2.0`
- **Example in Python:**

```
price = 19.99
temperature = -12.5
pi = 3.14159
```

3. String (`str`)

- Represents sequences of characters, used to store text.
- Strings are enclosed in either single quotes (`'...'`) or double quotes (`"..."`).
- Examples: `"Hello, World!"`, `'Python'`, `"123"`
- **Example in Python:**

```
name = "Alice"
greeting = 'Hello, World!'
number_as_text = "123"
```

4. Boolean (`bool`)

- Represents binary values, either `True` or `False`.
- Used for conditional logic (e.g., `if` statements).
- **Example in Python:**

```
is_active = True
has_permission = False
```

5. List (`list`)

- Represents an ordered collection of items, which can be of different data types.
- Lists are mutable, meaning they can be modified after creation.
- Examples: `[1, 2, 3]`, `['apple', 'banana', 'cherry']`, `[1, "text", 3.14]`
- **Example in Python:**

```
numbers = [1, 2, 3, 4, 5]
```

```
fruits = ["apple", "banana", "cherry"]
mixed = [1, "hello", 3.14, True]
```

6. Tuple (`tuple`)

- Represents an ordered collection of items, similar to lists, but tuples are immutable (cannot be modified).
- Examples: `(1, 2, 3)`, `("a", "b", "c")`, `(1, "hello", 3.14)`
- **Example in Python:**

```
coordinates = (10, 20)
colors = ("red", "green", "blue")
```

7. Dictionary (`dict`)

- Represents a collection of key-value pairs, like a mapping from one set of things to another.
- Each key is unique, and dictionaries are mutable.
- Examples: `{'name': 'Alice', 'age': 25}`, `{1: 'one', 2: 'two'}`
- **Example in Python:**

```
person = {
    "name": "Alice",
    "age": 25,
    "city": "New York"
}
```

8. Set (`set`)

- Represents an unordered collection of unique items.
- Sets are mutable and useful for removing duplicates or performing set operations like union and intersection.
- Examples: `{1, 2, 3}`, `{"apple", "banana", "cherry"}`
- **Example in Python:**

```
unique_numbers = {1, 2, 3, 4, 5}
fruits = {"apple", "banana", "cherry"}
```

9. NoneType (`None`)

- Represents a null or no-value.
- Used to indicate the absence of a value or a null value.
- **Example in Python:**

```
result = None
```

Summary

Python's flexibility allows you to define different types of variables depending on the data you need to represent. This variety of types enables you to create complex data structures, manage different kinds of information, and use efficient operations that match the type's characteristics.

Example: Asking for Name and Age:

```
# Step 1: Take user's name
name = input("What is your name? ")

# Step 2: Take user's age
age = input("How old are you? ")

# Step 3: Display the name and age
print("Hello, " + name + "! You are " + age + " years old.")
```

Explanation:

- **`input()`**: This function reads input from the user as a string.
- **Assigning input to a variable**: The value entered by the user is stored in the variable (e.g., `user_input`).
- **`print()`**: Displays the value of the variable.

This method works for all basic inputs and displays them accordingly.

To input three values into three different variables and find the maximum number using Python, you can follow this process:

Example Code:

```
# Step 1: Input three values from the user

num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
num3 = float(input("Enter the third number: "))

# Step 2: Find the maximum value using the max() function

maximum = max(num1, num2, num3)

# Step 3: Display the maximum number

print("The maximum number is:", maximum)
```

Explanation:

1. **`input()`**: Takes input from the user. We use `float()` to convert the input into a float number (or use `int()` if you want integer input).
2. **`max()`**: A built-in Python function that returns the largest value among the given arguments.
3. **`print()`**: Outputs the maximum value.

Sample Output:

```
Enter the first number: 12
Enter the second number: 25
Enter the third number: 7
The maximum number is: 25
```

To find the largest value among three inputs using `if-else` statements in Python, you can follow this example:

Example Code:

```
# Step 1: Input three values from the user

num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
num3 = float(input("Enter the third number: "))

# Step 2: Use if-else statements to find the largest number

if num1 >= num2 and num1 >= num3:
    largest = num1
elif num2 >= num1 and num2 >= num3:
    largest = num2
else:
    largest = num3

# Step 3: Display the largest number

print("The largest number is:", largest)
```

Explanation:

1. **Input Values**: The `input()` function is used to take three numbers from the user.
2. **Comparison**: The `if-else` statements compare the three numbers:
    - First, it checks if `num1` is greater than or equal to both `num2` and `num3`.
    - If not, it checks if `num2` is greater than or equal to both `num1` and `num3`.
    - If neither of the above conditions is true, it assumes `num3` is the largest.
3. **Output**: The largest number is printed.

Sample Output:

```
Enter the first number: 12
Enter the second number: 25
Enter the third number: 7
The largest number is: 25
```

This method works by explicitly comparing all three numbers using `if-else` logic.

In Python, the colon (`:`) is used to define the start of a block of code following control structures such as `if`, `elif`, `else`, `for`, `while`, `def`, `class`, and other compound statements.

Why is the colon (`:`) used?

1. **Indicates the Start of a Code Block**:
   - After control statements like `if`, `elif`, and `else`, the colon tells Python that what follows is a block of code to execute if the condition is true.
2. **Indentation**:
   - Python relies on indentation (usually 4 spaces) to structure the code logically. The colon marks the end of the condition and the start of the indented block. This block is considered the "body" of the control structure.

Example Without Colon (Invalid Syntax):

```
if (num1 >= num2) and (num1 >= num3)  # Missing colon
    largest = num1  # This will cause a SyntaxError
```

Example With Colon (Correct Syntax):

```
if (num1 >= num2) and (num1 >= num3):  # Correct use of colon
    largest = num1  # Indented block of code
```

Explanation:

- **Without the colon**: Python doesn't know when the condition ends and the block of statements should begin, leading to a `SyntaxError`.
- **With the colon**: It clearly separates the condition from the subsequent indented block of code, making it part of the `if` statement.

Summary:

- **Colon (`:`)**: Marks the start of a block of code in control structures.
- **Indentation**: Python uses indentation to determine the block of code that belongs to the control structure, and the colon helps to clearly define when that block begins.