Python Basics - Overview

---

1. Python - Variables and Data Types

Variables in Python: In Python, variables are dynamically typed, meaning you don't need to explicitly define the type of a variable at the time of its creation. Python automatically assigns the type based on the value you assign to it.

```
# Example of variables with different data types
x = 10        # Integer
y = 3.14      # Float
z = "Python"  # String
```

- **Reassigning variables:** You can change the type of a variable by assigning a new value to it.

  ```
  x = 10       # Initially an integer
  x = "Hello"  # Now a string
  ```

Common Data Types:

- **Integers (`int`)**: Whole numbers without decimal points.

  ```
  x = 100
  ```

- **Floating-point numbers (`float`)**: Numbers with decimal points.

  ```
  pi = 3.14159
  ```

- **Strings (`str`)**: A sequence of characters enclosed in quotes.

  ```
  name = "Alice"
  ```

- **Booleans (`bool`)**: Logical values representing `True` or `False`.

  ```
  is_active = True
  ```

- **Lists (`list`)**: Ordered, mutable collection of elements.

  ```
  fruits = ["apple", "banana", "cherry"]
  ```

- **Tuples (`tuple`)**: Ordered, immutable collection of elements.

  ```
  coordinates = (10, 20, 30)
  ```

- **Dictionaries (`dict`)**: Unordered collection of key-value pairs.

```
person = {"name": "John", "age": 30}
```

---

2. Python - Conditions and Iterations

Conditional Statements: Conditional statements allow you to make decisions in your program based on certain conditions.

- **If-Else Statements:** The basic syntax of an if-else condition is:

```
if condition:
    # Code block if the condition is true
else:
    # Code block if the condition is false
```

Example:

```
x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")
```

- **Elif (Else If):** If there are multiple conditions, you can use `elif` to check additional conditions.

```
if x > 10:
    print("x is greater than 10")
elif x == 10:
    print("x is equal to 10")
else:
    print("x is less than 10")
```

Looping (Iteration):

- **For Loop:** It is used to iterate over a sequence (e.g., list, tuple, or string).

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

- **While Loop:** It repeats a block of code as long as a condition is `True`.

```
count = 0
while count < 5:
    print(count)
    count += 1  # Increment to avoid infinite loop
```

---

3. Python - Type Casting and Exceptions

Type Casting: Type casting is the process of converting one data type to another. Python allows both implicit and explicit casting.

- **Implicit Casting (Automatic Type Conversion):** Python automatically converts one type to another if possible.

```
# Implicit casting
x = 10  # Integer
y = 2.5  # Float
result = x + y  # The integer is implicitly converted to a float,
resulting in 12.5
print(result)
```

- **Explicit Casting (Manual Type Conversion):** You can convert a value to a specific type using functions like `int()`, `float()`, and `str()`.

```
x = 3.14
y = int(x)  # Explicit casting from float to int
print(y)  # Output: 3
```

Exceptions: Exceptions are errors that occur during program execution. You can handle exceptions using `try`, `except`, and `finally` blocks to ensure that your program doesn't crash unexpectedly.

```
try:
    result = 10 / 0  # Attempt to divide by zero
except ZeroDivisionError:
    print("Cannot divide by zero.")
finally:
    print("Execution completed.")
```

- **Handling multiple exceptions:** You can handle multiple specific exceptions with multiple `except` blocks.

```
try:
    num = int(input("Enter a number: "))
    print(10 / num)
except ValueError:
    print("Invalid input, please enter an integer.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
```

---

4. Python - Functions and Built-in Functions

Functions: A function is a block of reusable code designed to perform a specific task. Functions are defined using the `def` keyword.

```
# Defining a function
def greet(name):
    return f"Hello, {name}!"

# Calling a function
message = greet("Alice")
print(message)  # Output: Hello, Alice!
```

- **Arguments and Return Values:** Functions can take arguments and return values. If no value is returned, Python returns `None` by default.

```
def add(a, b):
    return a + b

result = add(10, 20)
print(result)  # Output: 30
```

- **Default Arguments:** You can specify default values for function parameters.

  ```
  def greet(name="Guest"):
      return f"Hello, {name}!"
  print(greet())       # Output: Hello, Guest!
  print(greet("Bob"))  # Output: Hello, Bob!
  ```

Built-in Functions: Python offers a wide range of built-in functions, such as:

- `len()`: Returns the length of an object (string, list, etc.).

  ```
  python
  Copy code
  text = "Python"
  print(len(text))  # Output: 6
  ```

- `max()`: Returns the largest item in an iterable.

  ```
  numbers = [10, 20, 30]
  print(max(numbers))  # Output: 30
  ```

---

5. Python - Data Structures

Python provides powerful built-in data structures that allow you to store and manipulate data.

- **Lists:** A list is a mutable, ordered collection of items.

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")  # Adding a new item to the list
fruits[1] = "blueberry"  # Modifying an item in the list
```

- **Tuples:** A tuple is similar to a list but is immutable.

```
coordinates = (10, 20, 30)
# coordinates[1] = 15  # Error: Tuples are immutable
```

- **Dictionaries:** A dictionary stores key-value pairs. It is unordered.

```
person = {"name": "Alice", "age": 25}
print(person["name"])  # Output: Alice
person["age"] = 26     # Updating value
```

- **Sets:** A set is an unordered collection of unique elements.

```
numbers = {1, 2, 3, 4}
numbers.add(5)  # Adding an element
```

---

6. Python - Classes and Inheritance

Classes: A class is a blueprint for creating objects. Each object is an instance of a class.

```
# Defining a class
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def bark(self):
        print(f"{self.name} says Woof!")

# Creating an object (instance)
dog1 = Dog("Buddy", "Golden Retriever")
dog1.bark()  # Output: Buddy says Woof!
```

Inheritance: Inheritance allows a new class to inherit methods and attributes from an existing class.

```
# Base class
class Animal:
    def speak(self):
        print("Animal speaks")

# Derived class (inherits from Animal)
class Dog(Animal):
    def bark(self):
        print("Woof!")

dog = Dog()
dog.speak()  # Output: Animal speaks (inherited method)
dog.bark()   # Output: Woof! (method from Dog class)
```

- **Overriding Methods:** A derived class can override methods of the base class.

  ```
  class Cat(Animal):
      def speak(self):
          print("Meow!")

  cat = Cat()
  cat.speak()  # Output: Meow! (overridden method)
  ```

- **Constructor (`__init__`) in Inheritance:** When creating an object of a subclass, you can call the parent class's constructor using `super()`.

  ```
  class Bird(Animal):
      def __init__(self, species):
          super().__init__()
          self.species = species

  bird = Bird("Parrot")
  print(bird.species)  # Output: Parrot
  ```