Python Basics: Variables, Numbers, Strings

1. Variables in Python:

Variables in Python are used to store data, and they don't need explicit declaration of type. Python automatically assigns the data type based on the value.

Example:

```
name = "John"        # String
age = 25             # Integer
height = 5.9         # Float
is_student = True    # Boolean


print(name, age, height, is_student)
```

2. Numbers in Python:

Python supports integers, floats (decimal numbers), and complex numbers.

Integer Example:

```
x = 10
y = 20
sum_result = x + y
print(f"The sum of {x} and {y} is {sum_result}")


Float Example:
pi = 3.14159
radius = 5.0
circumference = 2 * pi * radius
```

```
print(f"The circumference of the circle is {circumference}")
```

Complex Number Example:

```
z1 = 2 + 3j
z2 = 4 + 5j
z_sum = z1 + z2
print(f"The sum of complex numbers {z1} and {z2} is {z_sum}")
```

3. Strings in Python:

Strings are sequences of characters enclosed in single or double quotes.

Example of basic string operations:

```
greeting = "Hello, World!"
name = "Alice"
message = greeting + " " + name
print(message)

String length:
length = len(greeting)
print(f"The length of the greeting is {length} characters")

Accessing characters in a string:
first_letter = greeting[0]
print(f"The first letter is {first_letter}")

String slicing:
hello_part = greeting[0:5]
```

```
print(f"Sliced string: {hello_part}")
```

4. String Methods:

Python provides many built-in string methods for string manipulation.

Examples:

```
sample = "  Hello, Python!  "


Strip whitespace:

trimmed = sample.strip()

print(f"Trimmed string: '{trimmed}'")


Convert to uppercase:

uppercase = trimmed.upper()

print(f"Uppercase string: {uppercase}")


Find a substring:

position = sample.find("Python")

print(f"The word 'Python' starts at position: {position}")


Replace substring:

new_string = sample.replace("Python", "World")

print(f"Replaced string: {new_string}")
```

5. Working with Numbers and Strings Together:

Python allows you to work with both numbers and strings in a single expression.

Python Basics: Lists, Dictionaries, Tuples, If Condition, For Loop

1. Lists in Python:

Lists are ordered, mutable collections of elements.

Example:

```
fruits = ["apple", "banana", "cherry"]
print(fruits)
```

# Adding elements

```
fruits.append("orange")
print(fruits)
```

# Accessing elements

```
first_fruit = fruits[0]
print(f"The first fruit is {first_fruit}")
```

2. Dictionaries in Python:

Dictionaries are collections of key-value pairs.

Example:

```
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}
print(person)
```

```
# Accessing a value
name = person["name"]
print(f"The person's name is {name}")

# Adding a key-value pair
person["email"] = "john@example.com"
print(person)
```

3. Tuples in Python:

Tuples are ordered, immutable collections of elements.

Example:

```
coordinates = (10, 20)
print(f"Coordinates: {coordinates}")
```

# Accessing tuple elements

```
x = coordinates[0]
y = coordinates[1]
print(f"x: {x}, y: {y}")
```

4. If Conditions in Python:

If conditions are used for decision making.

Example:

```
age = 20
if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")

# Multiple conditions with elif
score = 85
if score >= 90:
    print("A grade")
elif score >= 80:
    print("B grade")
else:
    print("C grade")
```

5. For Loops in Python:

For loops are used to iterate over sequences.

Example:

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(f"I like {fruit}")

# Iterating over a range of numbers
for i in range(5):
    print(f"Number: {i}")
```

Example:

```
name = "Bob"

age = 30

message = f"Hello {name}, you are {age} years old."

print(message)
```

Functions, Modules,
File Handling, Exception Handling

1. Functions in Python

A function is a block of reusable code that performs a specific task.

Example: Basic function

```
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))  # Output: Hello, Alice!
```

Example: Function with default arguments

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

```
print(greet())  # Output: Hello, Guest!
```

Example: Function with variable-length arguments (*args, **kwargs)

```
def add_numbers(*args):
    return sum(args)

print(add_numbers(1, 2, 3))  # Output: 6
```

2. Modules in Python

A module is a file containing Python code. It may contain functions, classes, or variables.

Example: Creating and using a module

Create a file my_module.py:

```
# my_module.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

Use the module in another Python script:

```
import my_module

print(my_module.add(10, 5))       # Output: 15
print(my_module.subtract(10, 5))  # Output: 5
```

Example: Using standard library modules

```
import math

print(math.sqrt(16))  # Output: 4.0
```

3. Reading Files in Python

Python provides several ways to read files. The most common is to use the open() function.

Example: Reading a file line-by-line

```
with open('example.txt', 'r') as file:
    for line in file:
        print(line.strip())  # strip() removes any trailing newline characters
```

Example: Reading the entire file at once

```
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
```

4. Writing Files in Python

You can also use the open() function to write to files. The w mode is for writing, and the a mode is for appending.

Example: Writing to a file

```
with open('output.txt', 'w') as file:
    file.write("Hello, World!\n")
```

Example: Appending to a file

```
with open('output.txt', 'a') as file:
    file.write("This is a new line.\n")
```

5. Exception Handling in Python

Exception handling in Python is done with try, except, and optionally finally blocks.

Example: Basic try-except block

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Example: Handling multiple exceptions

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ValueError:
    print("Invalid input! Please enter a number.")
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Example: finally block

The finally block is always executed, whether an exception occurs or not.

```
try:
    file = open("example.txt", "r")
    content = file.read()
except FileNotFoundError:
    print("File not found!")
finally:
    file.close()  # Always executed
```

Python Basics: Classes and Objects

1. Classes in Python:

A class is a blueprint for creating objects. It defines attributes (variables) and methods (functions) that the objects created from the class can have.

Example of a simple class:

```
class Dog:
    # Class attribute
    species = "Canine"


    # Constructor method (called when object is created)
    def __init__(self, name, age):
        self.name = name  # Instance attribute
        self.age = age    # Instance attribute
```

```
    # Method for the class
    def bark(self):
        return f"{self.name} says woof!"


# Creating an object from the class
my_dog = Dog("Buddy", 5)
print(my_dog.name)  # Accessing attribute
print(my_dog.bark())  # Calling method
```

2. Objects in Python:

An object is an instance of a class. Each object can have different attribute values but shares the same methods defined in the class.

Example of creating multiple objects:

```
dog1 = Dog("Buddy", 5)

dog2 = Dog("Max", 3)
```

```
# Accessing object attributes
```

```
print(f"{dog1.name} is {dog1.age} years old.")

print(f"{dog2.name} is {dog2.age} years old.")
```

```
# Calling object methods
```

```
print(dog1.bark())

print(dog2.bark())
```

3. Class vs Instance Attributes:

- Class attributes are shared by all instances of a class.

- Instance attributes are specific to each object.

Example:

```
class Car:

    wheels = 4  # Class attribute



    def __init__(self, brand, model):

        self.brand = brand  # Instance attribute

        self.model = model  # Instance attribute



car1 = Car("Toyota", "Camry")
```

```
car2 = Car("Honda", "Civic")
```

# Accessing class and instance attributes

```
print(car1.wheels)  # Both cars have the same number of wheels (class attribute)

print(car2.wheels)

print(car1.brand)  # Each car has its own brand and model (instance attributes)

print(car2.brand)
```

4. Inheritance in Python:

Inheritance allows a class to inherit attributes and methods from another class.

Example:

```
class Animal:

    def __init__(self, name):

        self.name = name


    def speak(self):

        raise NotImplementedError("Subclass must implement abstract method")


class Cat(Animal):

    def speak(self):

        return f"{self.name} says meow!"
```

```
# Creating an object of the subclass

my_cat = Cat("Whiskers")

print(my_cat.speak())
```

5. Encapsulation in Python:

Encapsulation restricts direct access to some of an object's components (usually through private variables).

Example:

```
class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.__balance = balance  # Private attribute

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Insufficient funds")

    def get_balance(self):
        return self.__balance

account = BankAccount("Alice", 1000)
account.deposit(500)
account.withdraw(300)
print(f"Balance: {account.get_balance()}")
```

6. Polymorphism in Python:

Polymorphism allows different classes to be treated the same way through a common interface.

Example:

```
class Bird:
    def speak(self):
        return "Tweet!"

class Dog(Animal):
    def speak(self):
        return f"{self.name} says woof!"

# Both classes have a speak() method, and we can use them interchangeably
animals = [Bird(), Dog("Buddy")]
for animal in animals:
    print(animal.speak())
```