PYTHON CONCEPTS FROM ZERO TO HERO

Python is a versatile, high-level programming language known for its simplicity and readability, making it an excellent choice for both beginners and experienced programmers. Below, I'll cover some fundamental Python concepts, ranging from basic to advanced topics:

1. **Basic Concepts**

Variables and Data Types
- **Variables**: Containers for storing data values. In Python, you don't need to declare the variable type explicitly.
  ```python
  x = 5       # integer
  y = 3.14    # float
  name = "Alice"  # string
  is_active = True  # boolean
  ```

- **Data Types**: Python has several built-in data types:
  - **Integers** (`int`): Whole numbers, e.g., `5`, `-2`
  - **Floats** (`float`): Decimal numbers, e.g., `3.14`, `-0.5`
  - **Strings** (`str`): Sequence of characters, e.g., `"hello"`
  - **Booleans** (`bool`): Represents `True` or `False`
  - **NoneType**: Represents the absence of a value (`None`)

Basic Operators
- **Arithmetic Operators**: `+`, `-`, `*`, `/`, `//` (floor division), `%` (modulus), `**` (exponentiation)
  ```python
  sum = 4 + 5   # 9
  difference = 10 - 6  # 4
  product = 7 * 3   # 21
  quotient = 8 / 2   # 4.0
  floor_div = 7 // 2  # 3
  modulus = 7 % 2  # 1
  power = 2 ** 3  # 8
  ```

- **Comparison Operators**: `==`, `!=`, `>`, `<`, `>=`, `<=`
  ```python
  print(5 == 5)  # True
  print(3 != 4)  # True
  print(7 > 2)   # True
  ```

- **Logical Operators**: `and`, `or`, `not`
  ```python
  ```

PYTHON CONCEPTS FROM ZERO TO HERO

```
print(True and False)  # False
print(True or False)   # True
print(not True)        # False
```

- **Assignment Operators**: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `**=`, `//=`

```python
x = 5
x += 3  # x = x + 3
print(x)  # 8
```

Conditional Statements

- **`if`, `elif`, `else`**: Control the flow of execution based on conditions.

```python
age = 18
if age >= 18:
    print("You are an adult.")
elif age > 12:
    print("You are a teenager.")
else:
    print("You are a child.")
```

Loops

- **`for` Loop**: Iterates over a sequence (like a list, tuple, or string).

```python
for i in range(5):
    print(i)  # Outputs 0, 1, 2, 3, 4
```

- **`while` Loop**: Repeats as long as a condition is `True`.

```python
count = 0
while count < 5:
    print(count)
    count += 1
```

2. **Data Structures**

Lists

- **Lists**: Ordered, mutable collections of items.

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

PYTHON CONCEPTS FROM ZERO TO HERO

```
fruits.append("orange")
print(fruits)  # ['apple', 'banana', 'cherry', 'orange']
```

Tuples
- **Tuples**: Ordered, immutable collections of items.
```python
coordinates = (10, 20)
print(coordinates[0])  # 10
```

Sets
- **Sets**: Unordered collections of unique items.
```python
unique_numbers = {1, 2, 3, 4}
unique_numbers.add(5)
print(unique_numbers)  # {1, 2, 3, 4, 5}
```

Dictionaries
- **Dictionaries**: Unordered collections of key-value pairs.
```python
student = {"name": "John", "age": 21, "courses": ["Math", "Science"]}
print(student["name"])  # John
```

3. **Functions**

- **Defining Functions**: Reusable blocks of code defined with the `def` keyword.
```python
def greet(name):
    return f"Hello, {name}!"

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

- ****Lambda Functions****: Small anonymous functions defined with the `lambda` keyword.
```python
add = lambda x, y: x + y
print(add(3, 4))  # 7
```

4. **Object-Oriented Programming (OOP)**

- ****Classes and Objects****: Define custom types and behaviors.

PYTHON CONCEPTS FROM ZERO TO HERO

```python
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        return "Woof!"

my_dog = Dog("Buddy", 3)
print(my_dog.name)  # Buddy
print(my_dog.bark())  # Woof!
```

- **Inheritance**: Create a new class that inherits methods and properties from another class.

```python
class Animal:
    def speak(self):
        return "Sound"

class Cat(Animal):
    def speak(self):
        return "Meow"

cat = Cat()
print(cat.speak())  # Meow
```

5. **Modules and Packages**

- **Modules**: Reusable pieces of code stored in files.

```python
# Importing a module
import math
print(math.sqrt(16))  # 4.0
```

- **Packages**: Collections of modules in directories with an `__init__.py` file.

6. **Error Handling**

- **Exceptions**: Handling errors with `try`, `except`, `finally`, and `raise`.

```python
try:
```

PYTHON CONCEPTS FROM ZERO TO HERO

```
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
finally:
    print("Execution finished.")
```

7. **File Handling**

- **Reading and Writing Files**: Use `open()` to read or write files.
```python
# Writing to a file
with open("example.txt", "w") as file:
    file.write("Hello, World!")

# Reading from a file
with open("example.txt", "r") as file:
    content = file.read()
    print(content)  # Hello, World!
```

8. **Advanced Topics**

List Comprehensions
- **List Comprehensions**: Concise way to create lists.
```python
squares = [x**2 for x in range(10)]
print(squares)  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Generators
- **Generators**: Functions that return an iterator using `yield`.
```python
def generate_numbers():
    for i in range(5):
        yield i

for number in generate_numbers():
    print(number)
```

Decorators
- **Decorators**: Modify the behavior of a function or method.
```python
def decorator_function(original_function):
```

```
    def wrapper_function():
        print("Wrapper executed this before {}".format(original_function.__name__))
        return original_function()
    return wrapper_function

@decorator_function
def display():
    print("Display function ran")

display()
```

9. **Python Standard Library and External Libraries**

- **Standard Library**: Built-in modules like `datetime`, `os`, `sys`, `json`, `csv`, etc.
  ```python
  import datetime
  print(datetime.datetime.now())  # Outputs the current date and time
  ```

- **External Libraries**: Installable via pip, such as `NumPy`, `Pandas`, `Requests`, and `Flask`.
  ```bash
  pip install requests
  ```

- **Example of Using an External Library**:
  ```python
  import requests

  response = requests.get("https://api.example.com/data")
  data = response.json()
  print(data)
  ```

10. **Best Practices**

- **PEP 8**: Follow the Python Enhancement Proposal 8 (PEP 8) for coding style.
- **Documentation**: Write clear docstrings for functions, classes, and modules.
- **Testing**: Use `unittest` or `pytest` for writing tests.
- **Version Control**: Use tools like Git for version control of your code.