UNIKASSEL
VERSITÄT

Getting Started with Python: Syntax, Functions, and Classes

Shahab A. Shojaeezadeh
shahab2710@gmail.com
github.com/shahab271069

October 28, 2024

Agenda

- Python Syntax Overview
- Data Structures: Lists, Dictionaries, Tuples
- Defining Functions
- Creating Classes
- Advanced Function Concepts
- Example and Best Practices

Python Syntax Overview

- Indentation is crucial (4 spaces recommended)
- Comments start with #
- Variables do not require explicit declaration
- Example:

Example

```
x = 10
print(x)
```

Data Structures: Lists

- Ordered and mutable collection
- Example:

List Example

```
my_list = [1, 2, 3, 4]
```

- Access elements via index:

Accessing List Elements

```
print(my_list[0])
```

Data Structures: Tuples

- Ordered and immutable collection
- Example:

Tuple Example

```
my_tuple = (1, 2, 3, 4)
```

- Access elements via index:

Accessing Tuple Elements

```
print(my_tuple[0])
```

Data Structures: Dictionaries

- Unordered collection of key-value pairs
- Example:

Dictionary Example

```
my_dict = "name": "Alice", "age": 25
```

- Access values via keys:

Accessing Dictionary Values

```
print(my_dict["name"])
```

Defining Functions

- Functions are defined using `def`
- Can take parameters and return values
- Example:

Function Definition

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

Calling Functions

- Functions are called by their name followed by parentheses
- Example:

Function Call

```
message = greet("Alice")
print(message)
```

Default Parameter Values

- Functions can have default parameter values
- Example:

Default Parameters

```
def greet(name="World"):
return f"Hello, name!"
```

Creating Classes

- Classes are defined using `class`
- Encapsulate data and functions
- Example:

Class Definition

```
class Dog:
def bark(self):
return "Woof!"
```

Instantiating Classes

- Create an object of a class by calling it
- Example:

Instantiate Class

```
my_dog = Dog()
print(my_dog.bark())
```

Class Constructors

- Use `__init__` to initialize attributes
- Example:

Class with Constructor

```
class Dog:
def __init__(self, name):
self.name = name
def bark(self):
elf.namereturn f"s says Woof!"
```

Example of Class Usage

- Create an instance with name:

Using the Class

```
my_dog = Dog("Rex")
print(my_dog.bark())
```

Advanced Function Concepts: Decorators

- Decorators are functions that modify other functions
- Example:

Using a Decorator

```
def my_decorator(func):
def wrapper():
print("Something is happening before the function is
called.")
func()
print("Something is happening after the function is
called.")
return wrapper
```

Method Overriding

- Subclass can provide a specific implementation of a method that is already defined in its superclass
- Example:

Method Overriding Example

```
class Animal:
def sound(self):
return "Some sound"
class Dog(Animal):
def sound(self):
return "Bark!"
```

List Comprehensions

- Concise way to create lists
- Example:

List Comprehension

```
squares = [x**2 for x in range(10)]
```

Lambda Functions

- Anonymous functions defined with `lambda`
- Example:

Lambda Function

```
add = lambda x, y: x + y
print(add(2, 3))
```

Error Handling

- Use `try` and `except` for error handling
- Example:

Error Handling

```
try:
print(1 / 0)
except ZeroDivisionError:
print("Cannot divide by zero!")
```

Best Practices

- Use descriptive names for functions and classes
- Keep functions small and focused
- Document your code with docstrings
- Follow PEP 8 style guide

Summary

- Python syntax is clean and readable
- Functions and classes allow for organized code
- Data structures like lists, tuples, and dictionaries are essential
- Many resources available for further learning

Questions?

Thank you! Any questions?