DE235 Soumyadeep Sinha
WEEK 2 - DAY 2 - 07-11-2024

PYTHON : Function / Packages

---

1. Functions in Python

- **Definition: Functions allow We to group code into reusable blocks. They are defined using the `def` keyword, followed by a function name, parameters in parentheses `( )`, and a colon `:`. Inside the function block, We can include code that will run when the function is called.**

Syntax:

```
def function_name(parameter1, parameter2, ...):
    # Function code
    return result  # Optional return statement
```

Example:

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

Calling Functions: We call functions by using their name followed by parentheses, including any necessary arguments. Python executes the function's code when it's called.

Types of Parameters:

Positional Arguments: Passed in the order they are listed in the function definition.

```
def add(x, y):
    return x + y
print(add(2, 3))  # Output: 5
```

Keyword Arguments: Specify arguments by name, which can be given in any order.

```
def profile(username, followers):
    print(f"Username: {username}, Followers: {followers}")
profile(username="John", followers=42)
```

Default Parameters: Provide default values, allowing the function to be called with fewer arguments.

```
def profile(username, followers=100):
    print(f"Username: {username}, Followers: {followers}")
profile("Alice")  # Output: Username: Alice, Followers: 100
```

Variable-Length Arguments:

- **`args`: Collects additional positional arguments as a tuple.**

- `*kwargs`: Collects additional keyword arguments as a dictionary.

```
def display(*args, **kwargs):
    print("Positional args:", args)
    print("Keyword args:", kwargs)

display(1, 2, 3, name="Alice", age=30)
# Output:
# Positional args: (1, 2, 3)
# Keyword args: {'name': 'Alice', 'age': 30}
```

Anonymous Functions (Lambdas):

- Syntax: `lambda parameters: expression`
- Commonly used for small functions without a `def` keyword.
- Example:

```
square = lambda x: x * x
print(square(5))  # Output: 25
```

Using with `filter`, `map`, and `reduce`:

- `filter(function, iterable)`: Filters elements based on a function.
- `map(function, iterable)`: Applies a function to each item in an iterable.
- `reduce(function, iterable)`: Reduces iterable to a single value (from `functools`).

```
from functools import reduce
lst = [1, 2, 3, 4]
squares = list(map(lambda x: x * x, lst))
even_numbers = list(filter(lambda x: x % 2 == 0, lst))
product = reduce(lambda x, y: x * y, lst)
```

2. Using `main()` as a Program Structure

- Including a `main()` function is optional but provides clarity and structure, especially for larger programs.
- Example:

```
def main():
    print("Main function is running")

if __name__ == '__main__':
    main()
```

This structure helps prevent parts of the code from running when imported into other modules.

3. Modules in Python

- Definition: A module is a Python file (ending in `.py`) containing functions, classes, and variables that We can import into other programs.

- **Creating a Module: Save reusable functions in a `.py` file. For example, create a file `calc.py` with:**

```
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y
```

Importing Modules:

- Basic Import: Use `import module_name` **to import a module.**
- Import with Alias: `import module_name as alias_name` **lets us rename the module for convenience.**
- Import Specific Function: `from module_name import function_name`**.**
- Import All: Use `from module_name import *` **to import all functions and variables.**

Python Module Search Path:

When importing, Python searches for modules in:

- **The current directory**
- **Directories in the `PYTHONPATH` environment variable**
- **Standard library directories**

View the search path with:

```
import sys
print(sys.path)
```

Useful Functions:

`dir()`: Lists all attributes (functions, classes, variables) in a module.

```
import calc
print(dir(calc))  # Lists add, subtract, etc.
```

reload( ) **(Python 3): Reloads an imported module, useful after making changes to a module file.**

```
from importlib import reload
reload(calc)
```

4. Package

A package is a collection of related modules in Python, helping organize code by bundling similar functionality together. Packages are usually organized with directories containing an `__init__.py` file, which marks the directory as a package.

```
# Directory structure
# mypackage/
# ├── __init__.py
```

```
# ├── module1.py
# └── module2.py

# Usage
from mypackage import module1, module2
```

5. Pip

Pip is Python's package manager for installing and managing libraries from the Python Package Index (PyPI). It simplifies working with third-party packages and dependencies.

Basic Pip Commands:

```
# Install a package
pip install package_name

# List installed packages
pip list

# Uninstall a package
pip uninstall package_name
```

6. Frameworks

A framework provides a structured environment for building applications by offering built-in components and tools. In Python, frameworks are popular in web development, data science, and machine learning.

Django

Django is a high-level web framework for building robust, scalable web applications quickly. It follows the Model-View-Template (MVT) architecture, emphasizing rapid development and clean design. Django includes tools for handling databases, user authentication, URL routing, and templates.

```
# Basic Django view example (views.py)
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello, Django!")
```

To create a Django project:

```
# Install Django
pip install django

# Create a Django project
django-admin startproject myproject

# Start the development server
python manage.py runserver
```

7. File Handling

File handling in Python involves reading and writing to files. Common modes include:

- `'r'` – read
- `'w'` – write (overwrites existing content)
- `'a'` – append
- `'r+'` – read and write

Example:

```
# Writing to a file
with open("example.txt", "w") as file:
    file.write("Hello, Python!")

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

Types of Files

- **Text Files (.txt, .csv): Store data in plain text, useful for readable data formats.**
- **Binary Files (.bin, .exe, .dat): Store data in binary format; more efficient for non-text data like images, executables, etc.**

8. Google Colab

Google Colab is a cloud-based Jupyter notebook environment provided by Google. It allows Python code execution in the cloud, with free access to GPUs and TPUs, making it ideal for machine learning and data science. Colab notebooks support file handling, data visualization, and integrating with Google Drive.

Features:

- **Code Execution: Write and execute Python code in cells.**
- **Collaboration: Share and edit notebooks with others in real-time.**
- **Access to GPUs/TPUs: Great for training machine learning models.**

Example: To mount Google Drive in Colab:

```
from google.colab import drive
drive.mount('/content/drive')
```