T04-01. Programming language

Require

- Estimate time: 6 day
- Raise any questions when problem occurs

Concept

Python

Overview

- Python is a high-level, interpreted programming language that was first released in 1991 by Guido van Rossum. It is designed to be easy to read and write, with a **simple and consistent syntax,** making it a popular choice for beginners and experts alike.
- Python is known for its **flexibility and versatility,** and is used in a wide range of applications, including web development, scientific computing, data analysis, artificial intelligence, and automation. It supports multiple programming paradigms, including object-oriented, functional, and procedural programming.
- Python has **a large standard library**, which provides a wide range of pre-built modules and functions that can be used to perform common tasks, such as working with files, networking, and regular expressions. It also has a large ecosystem of third-party packages and libraries, which extend its functionality even further.
- Python is an **interpreted language**, which means that it does not need to be compiled before it can be run. Instead, the Python interpreter reads and executes the code directly, making it easy to write and test code quickly.
- Python is available on a wide range of platforms, including Windows, macOS, and Linux, and is free and **open-source software**, meaning that it can be freely used, distributed, and modified by anyone.

Python tutorial

- https://www.tutorialspoint.com/python/index.htm
- https://www.python.org/
- https://www.w3schools.com/python/

Modular programming

- Modular programming is a programming paradigm that involves breaking a large program into smaller, more manageable modules, or units of code, that can be developed, tested, and maintained independently. These modules are designed to be self-contained, with well-defined interfaces that allow them to interact with other modules in a predictable way.

> *The goal of modular programming is to improve software quality, maintainability, and scalability by creating a more structured and organized codebase. By breaking a large program into smaller modules, it becomes easier to understand and modify the code, and changes made to one module are less likely to affect other parts of the program.*

Functions, **modules** and **packages** are all constructs in Python that promote code modularization.

Module

Module Overview

- Module is a file that contains Python code and can define functions, classes, and variables. A module can be imported into another Python file, allowing the code in the module to be reused in different parts of an application.
- To use a module in Python, you can import it using the `import` statement. For example, if you have a module named `mymodule.py`, you can import it using the following code:

```
import mymodule
```

- After importing the module, you can use the functions, classes, and variables defined in it. For example, if `mymodule` defines a function named `myfunction`, you can call it like this:

```
mymodule.myfunction()
```

> *You can also create your own modules by defining Python code in a file and saving it with a `.py` extension. Once you've created the module, you can import it into other Python files to reuse the code.*

Module Search Path

When you import a module in Python using the `import` statement, Python searches for the module in a specific order within a set of directories called the "module search path".

The module search path is a list of directories that Python searches for modules when you try to import them. The directories in the module search path are determined by a combination of the Python installation, the operating system, and the environment.

Here is the order in which Python searches for modules in the module search path:

1. The directory containing the script that is being executed.
2. The directories specified in the `PYTHONPATH` environment variable (if it is set).
3. The default system-wide directory for modules. This varies depending on the operating system and Python installation.
4. Any directories specified in the `sys.path` list.

You can view the module search path in Python by importing the `sys` module and examining its `path` variable. For example, you can run the following code to view the module search path:

```
import sys

print(sys.path)
```

Package

- In Python, a package is a collection of modules that are grouped together in a directory hierarchy. A package can contain one or more modules, as well as subpackages, which are themselves packages containing their own modules and subpackages.
- Packages in Python are used to organize and modularize code, making it easier to manage and reuse code across multiple projects. By grouping related modules together in a package, you can create a more organized and structured codebase.
- Packages in Python are structured using directories and files. Each package is represented by a directory that contains an `__init__.py` file, which can define package-level variables, functions, and classes. The `__init__.py` file is executed when the package is imported, and can be used to perform any necessary setup or initialization.
- To use a package in Python, you can import it using the `import` statement. For example, if you have a package named `mypackage`, you can import it using the following code

```
import mypackage
```

- After importing the package, you can access the modules and subpackages contained within it using dot notation.

For more detail: Module and package

Decorator

- A decorator is a special type of function that can modify the behavior of another function. A decorator is used to wrap or modify the original function without changing its code.
- Decorators are often used to add functionality to a function or to modify its behavior, such as logging information about when a function is called or checking whether a user is authorized to call the function.
- A decorator is defined using the `@` symbol followed by the name of the decorator function, which is placed immediately before the definition of the function it is decorating.

Example:

```
def my_decorator(func):
    def wrapper():
        print("Before function is called.")
        func()
        print("After function is called.")
    return wrapper

@my_decorator
def my_function():
    print("Function is called.")

my_function()
```

When you run this code, you will see the following output:

```
Before function is called.
Function is called.
After function is called.
```

In this example, the `my_decorator` function is defined as a decorator that takes a function `func` as an argument and returns a new function `wrapper` that adds the desired behavior to the original function. The `wrapper` function is defined to print a message before and after calling the original function `func`, which is passed as an argument to `my_decorator`. Finally, the decorator is applied to the `my_function` function using the `@my_decorator` syntax, which causes the `my_function` to be replaced with the wrapped `wrapper` function.

> *Python decorators are a powerful tool that can be used to add functionality to existing functions, and can be used to implement common design patterns such as memoization, authentication, and caching.*

For more detail: Decorator

API and Sanic Framework

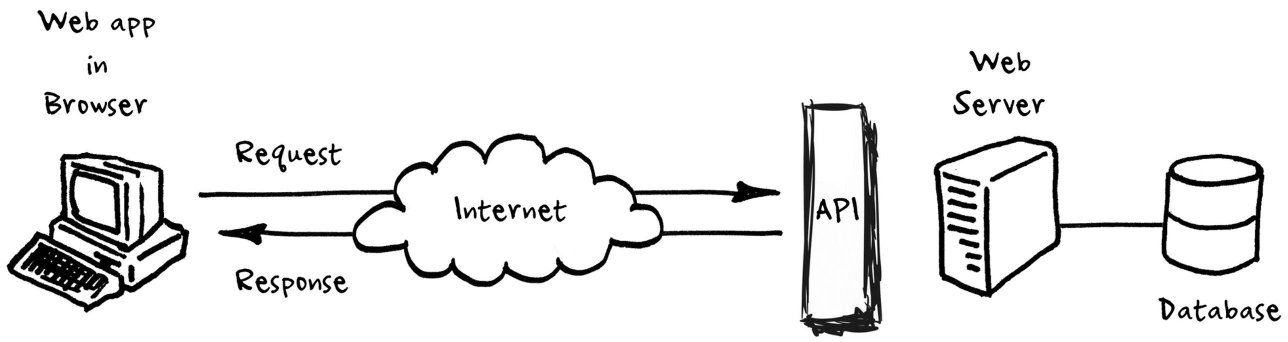
API

- API stands for "Application Programming Interface". In software development, an API is a set of protocols, routines, and tools for building software applications. APIs define how software components should interact with each other, allowing different software programs to communicate and exchange information.
- APIs can be used in a wide variety of contexts, including web development, mobile app development, and integration between different software systems. APIs can be used to access data, perform specific functions or operations, and provide access to various services or resources.
- APIs can take different forms, such as REST APIs (Representational State Transfer), SOAP APIs (Simple Object Access Protocol), and GraphQL APIs. REST APIs are a popular type of API that uses HTTP requests to retrieve and manipulate data, and is widely used in web and mobile app development.
- APIs have become a crucial part of modern software development, allowing developers to build complex applications and systems that can interact with a wide range of services and resources.

Reference: What is API

Restful API

- RESTful API stands for "Representational State Transfer API". It is a type of API that uses HTTP requests to retrieve and manipulate data. RESTful APIs are widely used in web and mobile app development.
- The main characteristics of RESTful APIs are:
  - Client-server architecture: The client and server are separate entities that communicate over the internet using HTTP requests and responses.



  - Stateless: Each request from the client to the server contains all the necessary information to complete the request, and the server does not maintain any state or session information between requests.
  - Uniform interface: The interface between the client and server is standardized and follows certain constraints, such as resource identification through URIs, manipulation of resources through HTTP methods, and representation of resources using a standard data format such as JSON or XML.
  - Cacheable: Responses from the server can be cached by the client or intermediary servers, to improve performance and reduce network traffic.
  - Layered system: A client can access a RESTful API through one or more intermediary servers, which can provide additional functionality or security features.
- RESTful APIs are widely used in web and mobile app development because they are simple, scalable, and can be used across a wide range of devices and platforms. They are used to retrieve and manipulate data from web services, databases, and other sources, and can be integrated with other APIs or services to build complex applications and systems.

Reference: What is RESTful API

Sanic Framework

Overview

- Sanic is a Python web framework that is designed to be fast and efficient. It is built on top of asynchronous Python libraries such as **asyncio** and **uvloop**, which allow it to handle a large number of requests with minimal resources.
- Sanic is similar in many ways to other Python web frameworks such as Flask and Django, but it offers some distinct advantages. These include:
  - High performance: Sanic is designed to be fast and efficient, making it a good choice for high-traffic web applications.
  - Asynchronous support: Sanic can handle asynchronous requests, which allows it to handle multiple requests simultaneously without blocking.
  - Lightweight: Sanic is a lightweight framework that has minimal dependencies and can be installed easily.
  - Flexible: Sanic is highly customizable, and it can be used to build a wide range of web applications, from simple APIs to complex web services.
- Sanic is an open-source framework that is maintained by a community of developers. It is available under the MIT license, which means that it can be used freely for both commercial and non-commercial projects.

Install and Get started

Instruction: https://sanic.dev/en/guide/getting-started.html

Sanic documentation

- Sanic Framework
- Tutorial
- https://www.geeksforgeeks.org/introduction-to-sanic-web-framework-python/

Exercise

- Prepare a presentation with the aim to:
  - Run simple **your own** python code with decorator, module & package
- Clone this API with Sanic Framework: Pet store API (1/2 query/command)
  - Clone data model
  - Build data table with model
  - Connect database with API
  - Build API with Sanic