Advanced Programming

Modules in Python

Ramaseshan Ramachandran

IMPORT STATEMENT

✦ Gains access to the code of one module in another

✦ Finds a Python module with a specified name to access the functions and data types defined within that module

✦ Two functions

  ✦ Searches for the module named in the import statement

  ✦ Binds the results of the search to the local scope

✦ Import programatically

```python
if __name__ == "__main__":
    import importlib
    my_math = importlib.import_module('math')
    print(my_math.__doc__)
    print(dir(my_math))
```

MODULES

- If the program gets longer, you may want to logically split it into several files for easier maintenance

- A function may also be used in several of your programs

- You want to use it without copying it

- Modules promote code organization and reusability.

- They group related functionalities into logical units

- They prevent global namespace pollution – names are local to the module

- A **module** is a file containing Python code that acts as a reusable building block

  - Self-contained library of functions, variables, and classes

Code organisation

Reusability

Maintainability

Namespace management

MODULE STRUCTURE

```
my_module/
├── __init__.py  # (Optional initialisation code)
├── function1.py
├── function2.py
```

__init__.py

- Marks a directory as a package

- Python interprets it as a package rather than a single module

- Package-level variables or constants

- Performing one-time initializations (e.g., connecting to databases)

- Setting up logging or configuration

Functions in Module Algorithms

```python
import math
def factorial(n) -> int:

    if not n >= 0:
        raise ValueError("n must be >= 0")
    if math.floor(n) != n:
        raise ValueError("n must be exact integer")
    if n+1 == n:  # catch a value like 1e300
        raise OverflowError("n too large")
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result

def fibonacci(n) -> int:
    if n <= 1:
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))

if __name__ == "__main__":
    print(factorial(5))
    print(fibonacci(15))
```

if _name_ == "_main_":

✦ _name_: A special built-in variable set to "_main_"

✦ Only executes when the file is run as a script

✦ Useful for tasks specific to standalone execution

✦ Advantages:

  ✦ Prevents unintended side effects when imported

  ✦ Creates cleaner and more modular code

SCOPE

✦ The scope determines the visibility and accessibility of variables within different parts code

✦ Scopes are implemented as dictionaries

✦ These dictionaries are called namespaces.

✦ Scope = {name:object}

✦ Follows Local, Enclosing, Global and built-in (LEGB) rule

✦ Global scope

```
global_var = 40

def my_function():
    loval_var = 100
    print(global_var)
if __name__ == '__main__':
    print(local_var)
    my_function()
```

Built-in Scope

Built-in functions and variables that are always accessible

  ○ Examples: print(), len(), True, False.

LEGB RULE

✦ When Python encounters a variable name

  ✦ Local: It checks if the variable is defined within the current block (function, loop, or conditional).

  ✦ Enclosing: If not found locally, it looks for it in enclosing blocks (outer functions/blocks).

  ✦ Global: If still not found, it checks the global scope of the module.

  ✦ Built-in: If not found globally, it checks the built-in scope

✦ Minimise global variable usage

✦ Use descriptive variable names

✦ Consider using classes to encapsulate related variables and functions, providing better control over their scope

Build-in Scope

Global Scope

Enclosing Scope
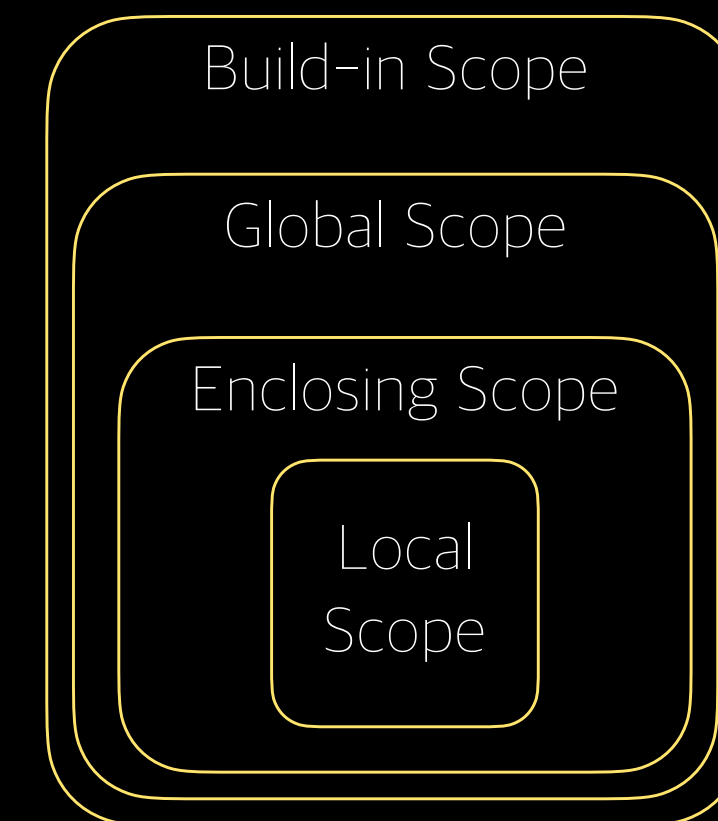
Local Scope

```python
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

if __name__ == '__main__':
    scope_test()
    print("In global scope:", spam)
```

```
After local assignment: test spam

After nonlocal assignment: nonlocal spam

After global assignment: nonlocal spam

In global scope: global spam
```