ComIT

Programming Paradigms

Programming Paradigms

- A “Programming Paradigm” is a high level way to organize your code and structure your software.
- There are many different paradigms, and programming languages can be described as supporting one or more programming paradigms.
- Python is a flexible language that can be used to code in a variety of different paradigms.

Modular Programming

- Modular Programming is a software design technique that emphasizes separating the functionality of a program into independent and interchangeable modules.
- A module is base unit of code that has everything it needs to “do it’s job.”
- This reflects an almost natural way that we often solve problems. A complex problem can be broken down into steps, and those steps can be broken down into sub-steps, etc.
- In modular programming, we would take each step of a problem solution and make it an independent module.

Modular Programming

- Modular Programming is one way a programming language can facilitate the construction of large scale applications.
- A large program is decomposed into a set of smaller programs, and those smaller programs are written is such a way as to allow for code reuse.
- Although now widespread and found in almost all programming languages, it took until the 1990s for the paradigm to catch on.
- Python uses modules as the primary unit of code organization!

Modular Programming – Benefits

- Modular code improves readability and comprehension by segmenting the code into clear, logical sections that each carry out a specific function.
- The use of smaller, self-contained modules simplifies management because alterations in one module generally do not affect others.
- Components, such as functions, from one module can be utilized in different sections of the program.
- All of the above make maintenance and scalability easier.

Modular Programming – Disadvantages

- Can increase complexity
- Can be time consuming and therefore more costly
- Modules that rely on each other can create circular dependencies. If care is not taken in code organization this can become cumbersome and time consuming to work around or fix.
- There may be performance issues, but this depends on the language, etc.

Modular Programming

- Until now we have been writing basic Python programs that, for the most part, only exist in a single file.
- This is fine for small programs, but quickly becomes an unsustainable way to organize code as the complexity of your software increases.
- If we want to develop more complex programs, we need to start spreading our source code across multiple files and directories.

Python Modules

- A Python module is a file containing Python definitions and statements.
- The file name is the module name with the suffix .py appended.
- Within a module, the module's name (as a string) is available as the value of the global variable __name__.
- When we want to access the variables, functions, and other attributes in a module, we need to import that module into our current **namespace**.
- We can use the *import* statement to import the module, and then access it's attributes with the "." operator.

Python Namespaces

- A namespace is a mapping from names to objects, and it is what is responsible for determining what identifiers are valid references to something and which ones are not.
  - This includes variable names, functions, class definitions, etc.
- Namespaces are created at different moments and have different lifetimes.
  - The namespace containing the built-in names is created when the Python interpreter starts up, and is never deleted.
  - The global namespace for a module is created when the module definition is read in.

Python Namespaces

- A namespace is a mapping from names to objects, and it is responsible for determining what identifiers are valid.
  - This includes variable names, functions, class definitions, etc.
- Namespaces are created at different moments and have different lifetimes:
  - The namespace containing the built-in names is created when the Python interpreter starts up, and is never deleted.
  - The global namespace for a module is created when the module definition is read in.
  - The local namespace for a function is created when the function is called, and deleted when the function returns or raises an exception that is not handled within the function.

Scope

- “Scope” is a term we use to describe when we have access to a namespace.
- In other words, when you try to reference a variable, or use a function, will Python know what you are trying to do?
- You can only reference identifiers when they are “in scope”, or when their namespace “exists”

Variable Scope

- A variable is only available from inside the region it is created.
- A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.
- A variable created in the main body of the Python code is a global variable and belongs to the global scope.
- If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.

Importing

- Importing is adding another module’s namespace to the current namespace.
- You can import at the beginning of a file (module) or at any other point during your program.
- You can import the entire module with the *import* statement, or you can select specific attributes to import with the *from <module> import <attribute>* syntax.

Packages

- Packages are ways of organizing modules and having them addressable with the "." notation.
  - For example, "animals.fish" would indicate that there is a "fish" module in the "animals" package.
- Directories are implicitly interpreted as packages called *namespace packages*. This sometimes breaks, and so adding an empty file called *__init__.py* to a directory with explicitly tell Python that the folder is a package.
- When importing a module, Python searches the directories that it "knows about" (those in *sys.path*).
- Packages can contain modules, or other sub packages

The Module Search Path

- A module search path is initialized when Python starts.
    - This module search path may be accessed at sys.path.
- The first entry in the module search path is **the directory that contains the input script.**
- The **PYTHONPATH** environment variable is often used to add directories to the search path. If this environment variable is found then the contents are added to the module search path.
- Built in Python modules are added after directories, and then finally any packages you have installed with your package manager.

Summary

- Python code can be organized into other files and folders
- Python file == “Module”
- Directory == “Package”
- Python will search for modules and packages based on the directories you tell it to look in.
- The first directory you “tell” it to look in is the parent directory of the script you are executing.
- You can add other directories in code or with environment variables.

Questions?