EPAM Python Software Engineer Training

Lesson 2: Python basics

Contents

1 Course

1.1 Python interpreter

Python can run in two modes: *interpreter* and *program*. Without arguments python is running in *interpreter* mode where one can pass any python commands interactively one-by-one. If some file or module is given as an argument python runs in *program* mode when it tries to execute module contents. An *interactive* mode is very useful for testing purposes when one wants to try a code snapshot quickly.

1.2 Modules and packages

A python project consists of one or more modules. Each python module is simply a file with `.py` extension containing python code. During execution each file is JIT-compiled into a `.pyc` file containing a byte-code that is being executed by an interpreter. For large projects it sometimes makes sense to pre-compile all modules. To find a module python is looking for the following files in directories listed in *PYTHONPATH*: `.so`, `.pyc`, `.py`. If a `.py` file is absent - a `.pyc` file is used. If both `.pyc` and `.py` files are present - a `.pyc` file might be regenerated based on a timestamp.

In case if a number of modules become large it makes sense to group them into packages. Python interpreter uses a naive directory structure for packaging. Each package is simply a directory containing a special file `__init__.py` and modules or other directories. Each module in that directory will be a member of that package. An `__init__.py` is a python module itself, however, there are rare cases when it contains any source code.

A module that is a start point for an application (the one that is passed as an argument into a python program) has a special name `__main__`. It is recommended to wrap all code that shall be executed at application start into this code block:

```
if __name__ == '__main__':
    # Here and application code goes.
```

1.3 Imports

In python there are 3 kinds of imports:

- absolute (**import package.module**);
- from-import (**from package import module**, **from package.module import attribute1, attribute2**);
- relative (**from . import module**, **from .module import attribute1**);

During an import python interpreter searches for a module specified inside import statement and loads its contents. By default python searches for modules in all directories specified in `PYTHONPATH`. However, it is possible to configure it to import files from anywhere (e.g. zip archive, database, network share, or RESTful web service, or even build them on the fly). All those techniques are not part of a training course.

1.4 Functions

Python is multi-paradigm language and a functional programming is one of its paradigms. Hence, every code instruction in python is a function call, even a class declaration. The simplest function is `def foo(): pass`. Each function returns something, by default it returns a `None` object. Function arguments can be positional and named. There are special arguments `*args` and `**kwargs` which allow a function to accept an arbitrary number of positional or named arguments accordingly.

All function arguments in python are passed by reference; the same applies to a function result. No implicit data coying appear during a function call.

1.5 Classes

In python there are *old-style* (until version `3.0`) and *new-style* classes (since version `2.2`). All modern applications use *new-style* classes that must be derived from `object`. The simplest *new-style* class is `class A(object): pass`.

In python everything is an object, hence, an instance of some class. Even a module, a function and a class itself are objects. Hence, they can be subclassed, created, and otherwise operated like objects.

Functions that are declared inside a class are methods. There are several kinds of methods: *instance methods*, *class methods*, and *static methods*. Each instance method accepts a class instance as the first attribute; each class method accepts a class as the first attribute. Instance and class methods can be called on both classes and class instances, while static methods can be called on classes only.

Instance methods can be either *bound* or *unbound*. Bound methods are those of a specific class instance and unbound methods belong to a class, not an instance. For example, consider the below code:

```
class A(object):

    def method(self):
        pass

a = A()

a.method()
A.method(a)
```

In both cases above the same method is called with the same arguments.

Each class in python has a special instance method `__init__` that is called instance initializer and is called right after an class instance was created. All arguments that are passed to a class constructor appear as arguments in that method.

2 Tasks

2.1 Packages

Create a package for this lesson. Create a module in it which prints `Hello World` and a current time.

Starting with this course each lesson shall be a separate package and each task shall be a separate module in it with the following naming convention: `lesson02.task01`.

Every module shall have its own meaningful doc-string and each public member of a module shall have a proper doc-string as well. A *lower_case* notation shall be used for all module members except for class names which shall follow a *CamelCase* notation.

2.2 Functions

Create a module containing the following functions:

- *factorial*: accepts one integer and prints its factorial (recursive);
- *my_args*: accepts an arbitrary number of arguments and prints them all;
- *harmony*: takes an arbitrary number of floats and prints their harmonic medium value;

A module shall call its functions with different argumets;

2.3 Classes

Create a module containing a class *MyNumberPrinter* which accepts a number in a constructor and contains the following instance methods:

- *me*: prints a number itself;
- *factorial*: prints a factorial of a number;
- *string*: prints a string concatenated with itself number times;
- *update*: modifies a number value and prints a new value;
- *time_in_past*: accepts a one letter string that is either of `s, m, h, d` and print a time that is a number of seconds, minutes, hours, or days in the past since now;

A module shall create several *MyNumberPrinter* instances showing its functionality.