author: Real Python John Sturtz
source: https://realpython.com/python-functional-programming/

Real Python - Functional Programming in Python: When and How to Use It

WW07.2 | Tuesday, February 15, 2022 | 04:12 PM

Functional programming is a paradigm where applies computational evaluation of functions. Although limited, Python has useful sets of functional methods built in the language.

Learning Objectives

- What the **functional programming** paradigm entails
- What it means to say that **functions** are **first-class citizens** in Python
- How to define **anonymous functions** with the `lambda` keyword
- How to implement functional code using `map()`, `filter()`, and `reduce()`

What Is Functional Programming?

The basis of functional programming are built with **pure function** that are functions that outputs values based on the arguments/inputs fed in without noticable "side effects" (modifies/changes values or states outside of local scope)

Benifits of a functional paradigm:

1. **High level**: Explicitly specify the results desired.
2. **Transparent**: Pure functions are dependent on only it's inputs and do not modify attributes in the environment; as a result, reduce debuging overhead
3. **Parallelizable**: Because pure functions only operate with what it is given, without side effects, it can be more easily ran in parallel with one another.

How Well Does Python Support Functional Programming?

Functional programming has the ability to

1. Take a function as an argument in another

```
def inner():
    print("I am function inner()!")

def outer(function):
    function()

outer(inner)
# I am function inner()!
```

3. Is able to return another function to its caller

Everything in Python is an object, and all objects, including functions, in python are of relitive equal stature.

Functions in Python are "**First-class citizens**", like strings and numbers, they can be passed around to other functions; for example, you can assign a function to a variable.

```
def func():
     print("I am function func()!")

func()
# I am function func()!

# Assign a function to a variable
another_name = func # Creates a new reference to func()
another_name()
# I am function func()!

"""Can display a function to console with print()"""
print("cat", func, 42)
# cat <function func at 0x7f81b4d29bf8> 42

"""Can include it as an element in a composit data object like a list"""
objects = ["cat", func, 42]
objects[1]
# <function func at 0x7f81b4d29bf8>
objects[1]()
# I am function func()!

"""Can use it as a dictionary key"""
d = {"cat": 1, func: 2, 42: 3}
d[func]
# 2
```