12 Functions

Key terms: `def return lambda`

Reading: Severance 4

Exercise: Write a program that dies with a RecursionError.

Compartmentalizing procedures with defined interfaces is nearly as old as programming itself; it was proposed by Grace Murray Hopper in 1952. Functions, or methods, are in Python declared using the keyword `def` and a collection of arguments. All calls are by reference and these arguments are normally untyped. Functions have their own scope, so names defined there do not exist elsewhere. They can return None or any other object, including collections.

```
# Defining a temperature conversion function:
def c2f(c):
 return 32+9*c/5
# Defining its inverse as an inline lambda function:
f2c = lambda f: 5*(f-32)/9
```

It's important to verify that functions operate properly; that for a given input, they produce the expected output. Usually, we would want to test many cases, and have them all be correct; this is called unit testing. In the absence of a formal testing framework the programmer might still want to make some checks:

```
# Verify that f2c and c2f agree about certain temperatures:
print(f2c(32)==0)
print(c2f(100)==212)
print(f2c(-40)==c2f(-40))
```

The two sorts of arguments of a functional call are positional and named. Positional ones occur in order, resembling a `tuple`; named arguments are more like a `dict`. In either case, default values (with `=`) makes it more possible for a function to serve both basic and advanced uses.

```
# Define and employ a helper function for annotating log entries.
# Note the combined positional and named arguments.
def annotate ( message, subsystem='unspecified' ):
 return '{}: {}.'.format(subsystem,message)
print ( annotate ( message="It is midnight" ) )
print ( annotate ( message='Irrigation commences',
 subsystem='garden' ) )
```

It bears mentioning that Python supports statically typed function definitions since version 3.5.[15] As an example, we can guarantee that a conversion function receive only numbers:

```
# A conversion that rejects nonnumeric arguments:
def c2f(c:float):
 return 32+9*c/5
```

Advanced function techniques like generators, decorators, and execution pools are covered in CS231, Advanced Python Programming.

Next, we will address regular expressions.

[15] `https://docs.python.org/3/library/typing.html`