Functions, Namespaces and Scopes, and Recap

YE Yuxiao

Tsinghua University

yeyuxiao@outlook.com

2017/10/20

Overview

1. Functions
   - Abstraction
   - Define a Function
2. Namespaces and Scopes
3. Recap

Abstraction

Abstraction is a key concept in computer programming.

- We often use the same or similar code over and over
- We may want to perform some action without worrying about the details of how it is done
- E.g. To get the sum of `l = [1, 2, 3, 4]`, we don't usually do `l[0] + l[1] + l[2] + l[3]`. Instead, we abstract the action of "adding up a sequence of numbers" into a function *sum()* and use `sum(l)`.

Functions follow the concept of abstraction, which allow us to compartmentalise bits of code and recall it when needed.

- Makes code easier to read
- Allows us to reuse code

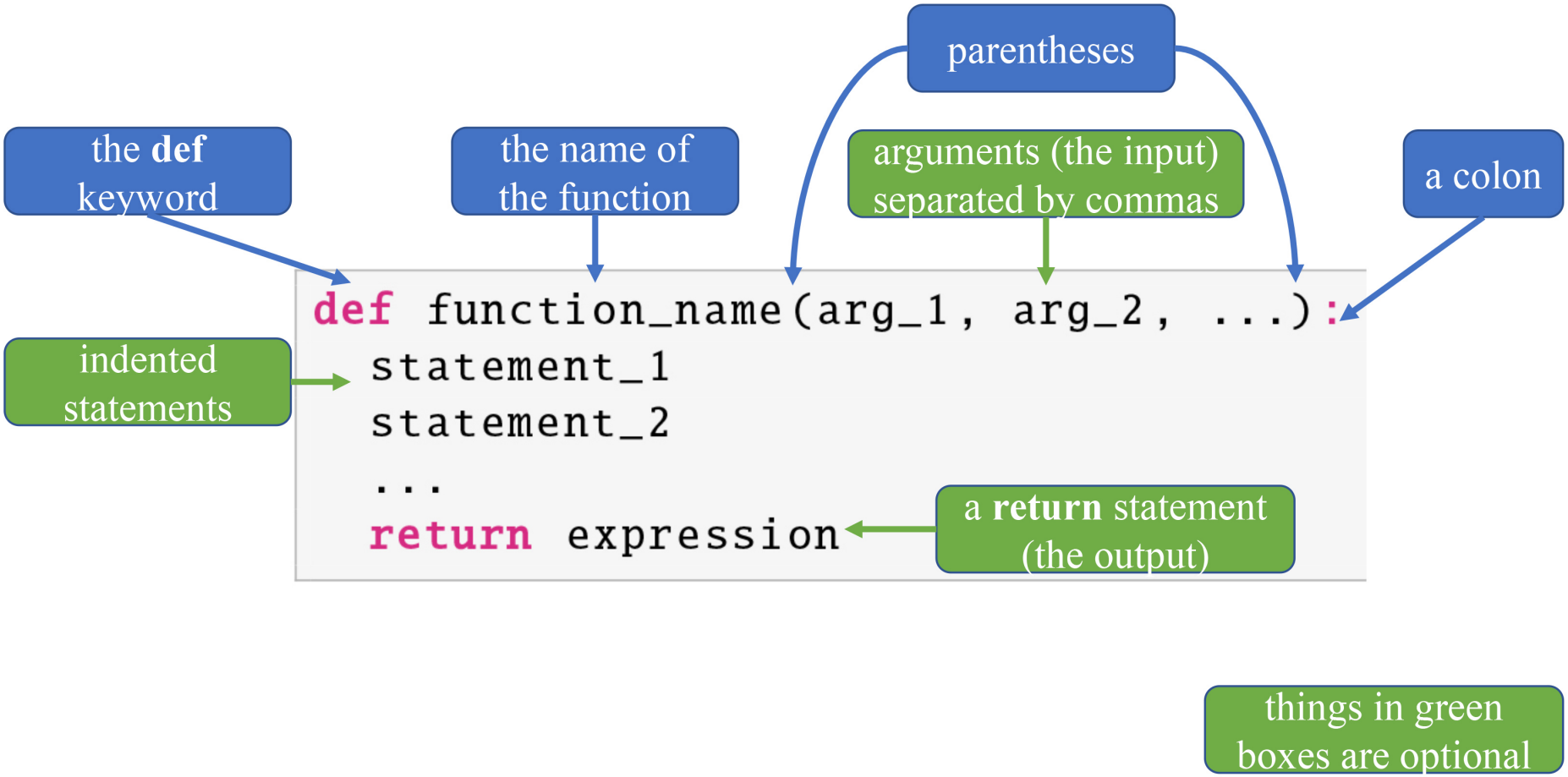
Define a Function

- If there wasn't a *sum()* function, we could write one:

```
> def my_sum(a_list):
    total = 0
    for i in a_list:
      total += i
    return total
```

Define a Function

- Defining a function in detail:



More Examples of Functions

- A function not taking any arguments:

```
> def give_me_5():
    return 5
```

- A function not returning any value:

```
> def say_hi(name):
    s = "Hi, " + name
    print (s)
```

Namespaces and Scopes

A **namespace** is a mapping from names (variables) to objects (their values)

- e.g. a = 5, s = "Hello world!"

A **scope** is part of a program where a namespace is directly accessible, i.e. you can refer to names directly

- A Python module creates a global scope
- Defining/calling a function creates a local namespace, who's scope is the duration of the function
- Names are resolved by searching the namespace associated by the innermost local scope first

Namespaces and Scopes

Names defined inside a function can't be resolved outside

```
> def say_hi(name):
      s = "Hi, " + name
      print (s)
> say_hi("Carlos")
Hi, Carlos
> s
NameError: name 's' is not defined
```

Namespaces and Scopes

Names defined in a global scope can be resolved in the local scope, if it hasn't been redefined in the local scope

```
> n = 1
> def plus1(m):
      m += n
      print (m)
> plus1(4)
  5
> def give_me_5():
      n += 4
      print (n)
> give_me_5()
  UnboundLocalError: local variable 'n' referenced before assignment
```

Recap

What we have learned so far:

- Data types: numbers, strings, lists, tuples, dictionaries
- Sequence operations: indexing and slicing
- Structured programs: loops, conditions, functions
- File I/O

Congratulations! You should now be able to write your own programs!

Exercise

Background: The Soundex algorithm (Odell and Russell, 1922; Knuth, 1973) is a method commonly used in libraries and older Census records for representing people' s names. It has the advantage that versions of the names that are slightly misspelled or otherwise modified (common, for example, in hand-written census records) will still have the same representation as correctly-spelled names. (e.g., Jurafsky, Jarofsky, Jarovsky, and Jarovski all map to J612).

Exercise

Write a FST (a function) to implement the Soundex algorithm, which is explained as follows:

a. Keep the first letter of the name, and drop all occurrences of non-initial a, e, h, i, o, u, w, y

b. Replace the remaining letters with the following numbers:

   b, f, p, v $\rightarrow$ 1
   c, g, j, k, q, s, x, z $\rightarrow$ 2
   d, t $\rightarrow$ 3
   l $\rightarrow$ 4
   m, n $\rightarrow$ 5
   r $\rightarrow$ 6

c. Replace any sequences of identical numbers , only if they derive from two or more letters that were adjacent in the original name, with a single number (e.g., 666 $\rightarrow$ 6)

d. Convert to the form Letter Digit Digit Digit (e.g., J612) by dropping digits past the third or padding with trailing zeros if necessary

Tip: Use what we've learned so far to solve this problem.