Functions and encapsulation

Wednesday, May 29, 2019 3:53 PM

Very often, we want to give a name to some concept of how to do something.

- … to remember a complicated set of steps.
- … to be able to repeat the steps somewhere else in the program.
- … to share the steps with other people.

There are lots of other reasons to make up a function.

The idea of a function

- Suppose we have something very clever. The following code removes duplicates from a list and sorts it into alphabetical order.

```
unique = sorted(list(set(things)))
```

- It would be good to remember this trick.
- So we might code it as a function:

```
def unique(things):
    return sorted(list(set(things)))
```

- Then we can say

```
stuff = unique(things)
```

- and stuff will contain the unique ones, in sorted order.

Some facts about functions

- Must be defined (using '`def`') before use.
- Stay defined once defined.
- Scope is all cells.

Encapsulation

- The process of putting a complex idea into a function is called *encapsulation*.
- In theory, you can forget the details and just remember the contract.
- In the function above, all you need to remember is that the one parameter has to be a list of strings.
- If that is true, the function works as expected.
- If that is not true, it might not work.

A rather complex example

- Suppose we are tracking debts of one person to another.
- Let's define the record ('Alva', 'Frank', 10) to mean "Alva owes Frank $10).
- Then we can make a list of debts, to wit:

```
debts = [("Alva", "Frank", 10),
         ("Fred", "George", 3),
         ("Amy", "George", 2),
         ("Frank", "Fred", 4),
         ("Frank", "Amy", 5)]
```

- Then we can ask questions about who owes who what.

First question: what people are represented.

- In this database, there are two columns that both represent people.

- `What people are represented?`
- `The following code computes this:`

```
people = set()
for d in debts:
    people.add(d[0])
    people.add(d[1])
sorted(list(people))
```

This is a bit subtle:

- A set automatically eliminates duplicates.
- So when I add someone twice, the person is only added once to the set.
- But sets can't be ordered.
- So I have to translate the `set` to a `list` and then I can sort it!

Making this into a function:

1. Determine the input variables: in this case, 'debts'.
2. Determine what the output should be: in this case, the final clause.
3. Encapsulate the code in a function with the input and output.

```
def people(debts):
    out = set()
    for d in debts:
        out.add(d[0])
        out.add(d[1])
    return sorted(list(out))
```

4. Call this as

```
plist = people(debts)
```

Another function: what is a person's balance?

- balance is what you're owed minus what you owe.
- Let's try this:

```
person = 'Frank'
balance = 0
for d in debts:
    if person == d[0]:  # person owes d[2]
        balance -= d[2]
    if person == d[1]:  # person is owed d[2]
        balance += d[2]
balance
```

Making this a function:

- Identify inputs: debts and p
- Identify output: balance
- Then we get:

```
def balance(debts, person):
    balance = 0
    for d in debts:
        if person == d[0]:  # person owes d[2]
            balance -= d[2]
        if person == d[1]:  # person is owed d[2]
            balance += d[2]
    return balance
```

Functions and mutability

- Mutability means the ability to be changed.
- Some objects are mutable inside a function when passed to it.
- Some objects are not.
- Notably,
    - Lists, sets, and dictionaries are mutable.

    - Numbers, tuples, and strings are not.
- This limits what you can do in a function.

Here are some simple functions that you can play with in the notebook for this unit:

```
def foo(bar):  # never changes the outside copy of
bar.
    bar = 1

def muck(thing):  # works if thing is a list.
    thing[0] = "kilroy was here"
```