Functions as Objects and Role of Objects in Python

Functions as Objects

In some languages, such as Python and Ruby, functions themselves are objects, i.e. instances of special classes. For example, functions built into the interpreter are instances of the `builtin_function_or_method` class:

```
>>> type(pow)
<class 'builtin_function_or_method'>
```

Similarly, as demonstrated above for the `deposit` function and method, user-defined functions are instances of `function`, and user-defined methods are instances of `method`:

```
>>> type(Account.deposit)
<class 'function'>
>>> type(tom_account.deposit)
<class 'method'>
```

By calling `mro` on the resulting types, we can see that they all inherit from `object`, just like the classes we defined above. (The names `builtin_function_or_method`, `function`, and `method` are not bound in the global namespace, so we cannot directly call `mro` on them.)

```
>>> [c.__name__ for c in type(pow).mro()]
['builtin_function_or_method', 'object']
>>> [c.__name__ for c in type(Account.deposit).mro()]
['function', 'object']
>>> [c.__name__ for c in type(tom_account.deposit).mro()]
['method', 'object']
```

Since functions are objects, they have attributes just like any other object. For example, all functions have a `__name__` attribute:

```
>>> pow.__name__
'pow'
```

Attributes can also be added to user-defined functions, and most existing attributes can be modified. Methods and built-in functions, however, don't allow adding or changing attributes.

```
>>> def my_pow(x, y):
        return pow(x, y)
>>> my_pow
<function my_pow at 0x7f77b2558270>
>>> my_pow.__name__ = "power"    # change attribute
>>> my_pow
<function power at 0x7f77b2558270>
>>> my_pow.value = 42   # new attribute
>>> my_pow.value
42
```

(Note: As of Python 3.3, functions have a `__qualname__` attribute that is used in many places instead of `__name__`. In the example above, it is the `__qualname__` attribute that must be changed instead of `__name__` to achieve the same behavior in Python 3.3.)

Function attributes allow us to store function-specific data with the function itself, without polluting the global namespace.

The Role of Objects

The Python object system is designed to make data abstraction and message passing both convenient and flexible. The specialized syntax of classes, methods, inheritance, and dot expressions all enable us to formalize the object metaphor in our programs, which improves our ability to organize large programs.

In particular, we would like our object system to promote a *separation of concerns* among the different aspects of the program. Each object in a program encapsulates and manages some part of the program's state, and each class statement defines the functions that implement some part of the program's overall logic.

Abstraction barriers enforce the boundaries between different aspects of a large program.

Object-oriented programming is particularly well-suited to programs that model systems that have separate but interacting parts. For instance, different users interact in a social network, different characters interact in a game, and different shapes interact in a physical simulation. When representing such systems, the objects in a program often map naturally onto objects in the system being modeled, and classes represent their types and relationships.

On the other hand, classes may not provide the best mechanism for implementing certain abstractions. Functional abstractions provide a more natural metaphor for representing relationships between inputs and outputs. One should not feel compelled to fit every bit of logic in a program within a class, especially when defining independent functions for manipulating data is more natural. Functions can also enforce a separation of concerns.

Multi-paradigm languages like Python allow programmers to match organizational paradigms to appropriate problems. Learning to identify when to introduce a new class, as opposed to a new function, in order to simplify or modularize a program, is an important design skill in software engineering that deserves careful attention.

Source : http://inst.eecs.berkeley.edu/~cs61A/book/chapters/objects.html#functions-as-objects