Functions & abstraction

Python Reference

Defining & calling functions

A function 函数 is a named block of code you can reuse. Define 定义 it with `def`, then call 调用 it by name.

```
def greet():
    print("Hello!")

greet()        # Hello!
greet()        # Hello!
```

- The code inside runs only when you call the function.

Return values

A function can return 返回 a value with `return`. The call then stands for that value.

```
def square(n):
    return n * n

print(square(5))          # 25
print(square(3) + 1)      # 10
```

- `return` ends the function at once. A function with no `return` gives `None`.

Parameters, arguments & scope

A parameter 形参 is the name in the `def`. An argument 实参 is the value you pass in.

```
def power(base, exp):         # base, exp are parameters
    return base ** exp

print(power(2, 3))            # 8  (2 and 3 are arguments)
```

A variable made inside a function is local 局部—it exists only there. That region is its scope 作用域.

```
def f():
    x = 10            # local to f
    return x

print(f())            # 10
# print(x) here would be an error: x is not defined outside f
```

Procedural abstraction

Procedural abstraction 过程抽象 means hiding details behind a name. You use a function by its name and what it does, not by how it works.

```
def area_of_rectangle(w, h):
    return w * h

print(area_of_rectangle(4, 5))   # 20
```

- A good function does one job, has a clear name, and avoids repeating code.

Modules & imports

A module 模块 is a file of ready-made functions. Bring one in with `import` 导入.

```
import random
random.seed(0)                # makes the result repeatable
print(random.randint(1, 6))  # a dice roll

import math
print(math.sqrt(16))          # 4.0
```