

Functions & abstraction

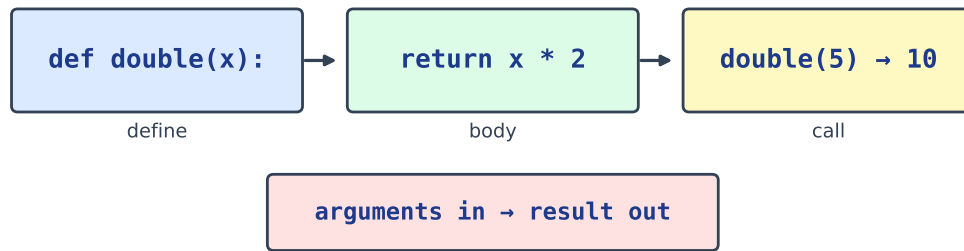
Python Reference

Defining & calling functions

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

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

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



Exam tip: `def` creates the function; `call` runs it; `return` sends a value back to the caller

def defines; call runs; return sends a value back

Return values

A function can return 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 is the name in the `def`. An argument 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 — it exists only there. That region is its scope .

```
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
```

A parameter can have a default value , used when the caller leaves it out:

```
def greet(name, greeting="Hello"):  
    return greeting + ", " + name  
  
print(greet("Mei"))      # Hello, Mei  
print(greet("Sam", "Welcome"))  # Welcome, Sam
```

Procedural abstraction

Procedural abstraction 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 is a file of ready-made functions. Bring one in with `import` .

```
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
```

Common mistakes

- A function returns `None` unless you write `return`. Printing is not the same as returning.
- Never use a mutable default like `def f(x=[])` — the same list is shared between all calls.
- A variable made inside a function is local and cannot be seen outside it.
- Run a function with `f()`; writing `f` on its own only names it.