SWCON104
Web & Python Programming

Function

Department of Software Convergence

경희대학교
KYUNG HEE UNIVERSITY

Today

- Review variables and computer memory
- Python-provided functions
- Defining your own functions
- Local variables
- Tracing function calls in the memory model
- Designing new functions

[Textbook]
Practical Programming
(An Introduction to Computer Science Using Python),
by Paul Gries, Jennifer Campbell, Jason Montojo,
The Pragmatic Bookshelf, 2017

Practice

- Practice_04_Functions

Variables

- Let's give a name to a value
  - X, species5618, degrees_celsius
  - 777obj(X), no-way(X), hello!(X)
- Assignment statement

```
>>> degrees_celsius = 26.0
```

- You can assign a new value to the existing variable

```
>>> degrees_celsius = 26.0
>>> degrees_celsius
26.0
>>> 9 / 5 * degrees_celsius + 32
78.80000000000001
>>> degrees_celsius / degrees_celsius
1.0
```

```
>>> degrees_celsius = 26.0
>>> 9 / 5 * degrees_celsius + 32
78.80000000000001
>>> degrees_celsius = 0.0
>>> 9 / 5 * degrees_celsius + 32
32.0
```

- Note that = means "assignment", not "equality"

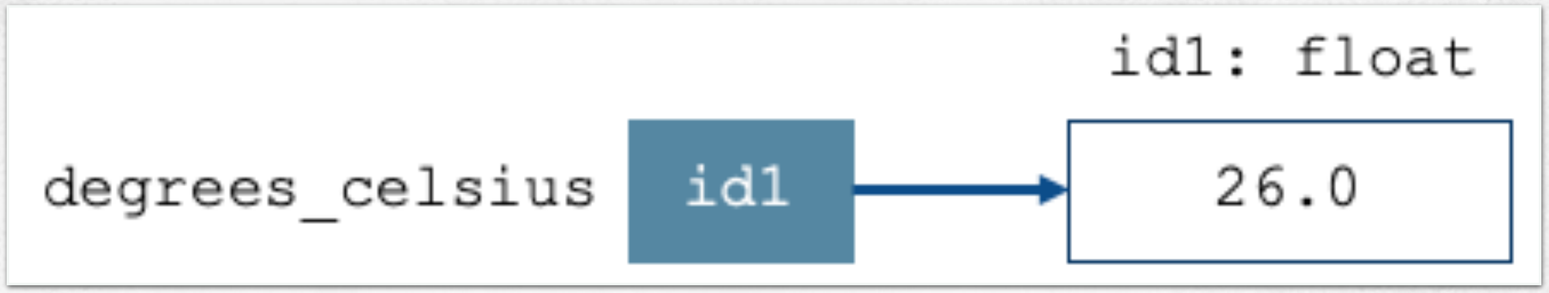
Variable

- Every location in the computer's memory has a memory address
- Object: a value (or thing) at a memory address with a type

26.0 **id1** **float**

- Variable contains the memory address of the object **degrees_celsius**

Values, variables, and computer memory



- Object: a value at a memory address with a type

  26.0 id1 float

- Variable contains the memory address of the object

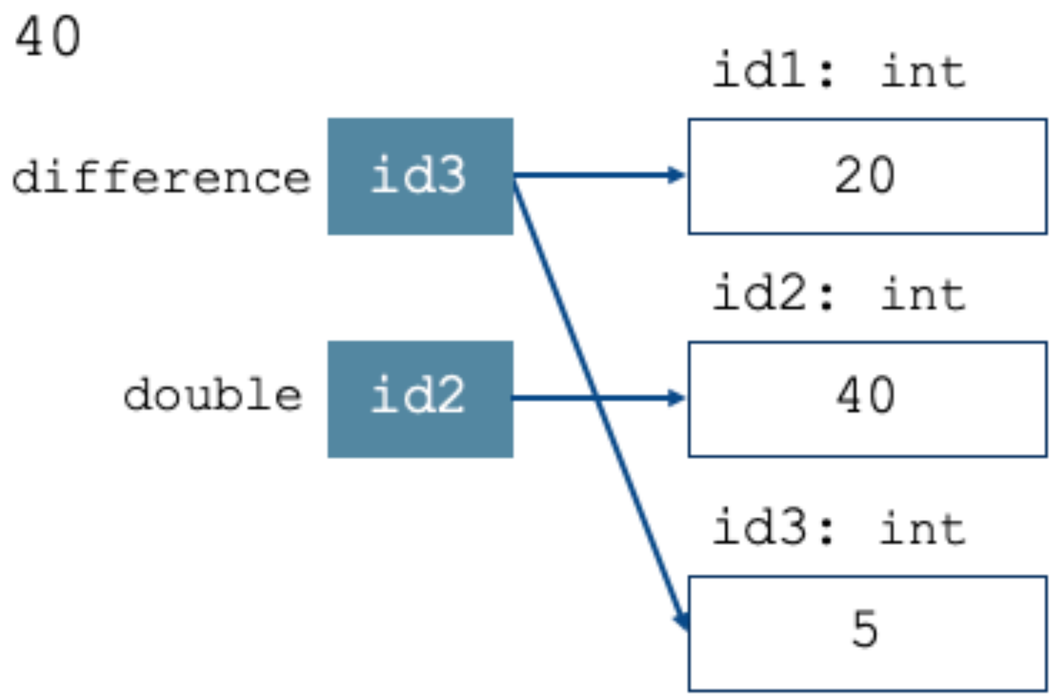
  `degrees_celsius`

  - Value 26.0 has the memory address id1.
  - The object at the memory address id1 has type float and the value 26.0
  - Variable degree_celsius contains the memory address id1.

Assignment statement

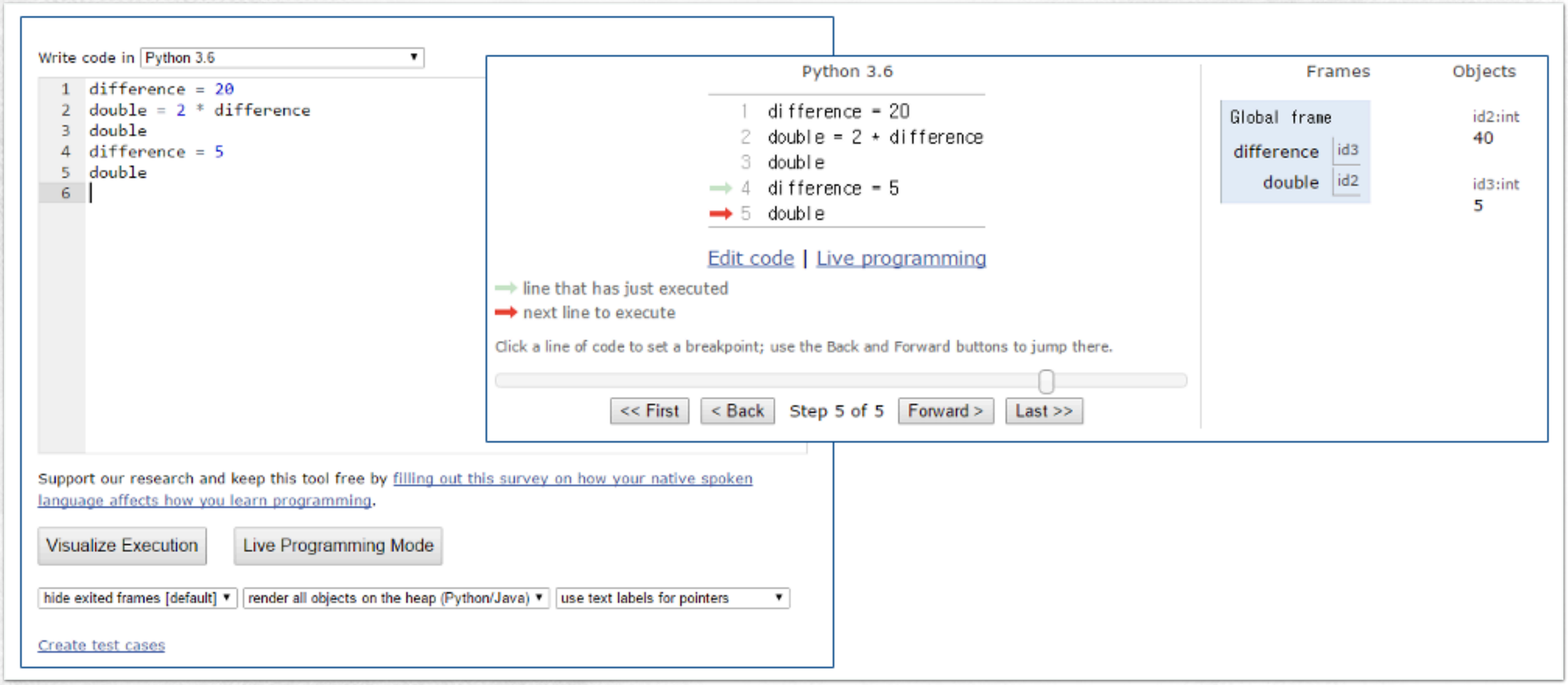
```
>>> degrees_celsius = 26.0 + 5
>>> degrees_celsius
31.0
```


```
>>> difference = 20
>>> double = 2 * difference
>>> double
40
>>> difference = 5
>>> double
40
```



Memory visualization

- http://pythontutor.com/visualize.html



Functions

- In mathematics

$$y = f(x) = x^2 + 3x + 2$$

$$f(2) = 12$$

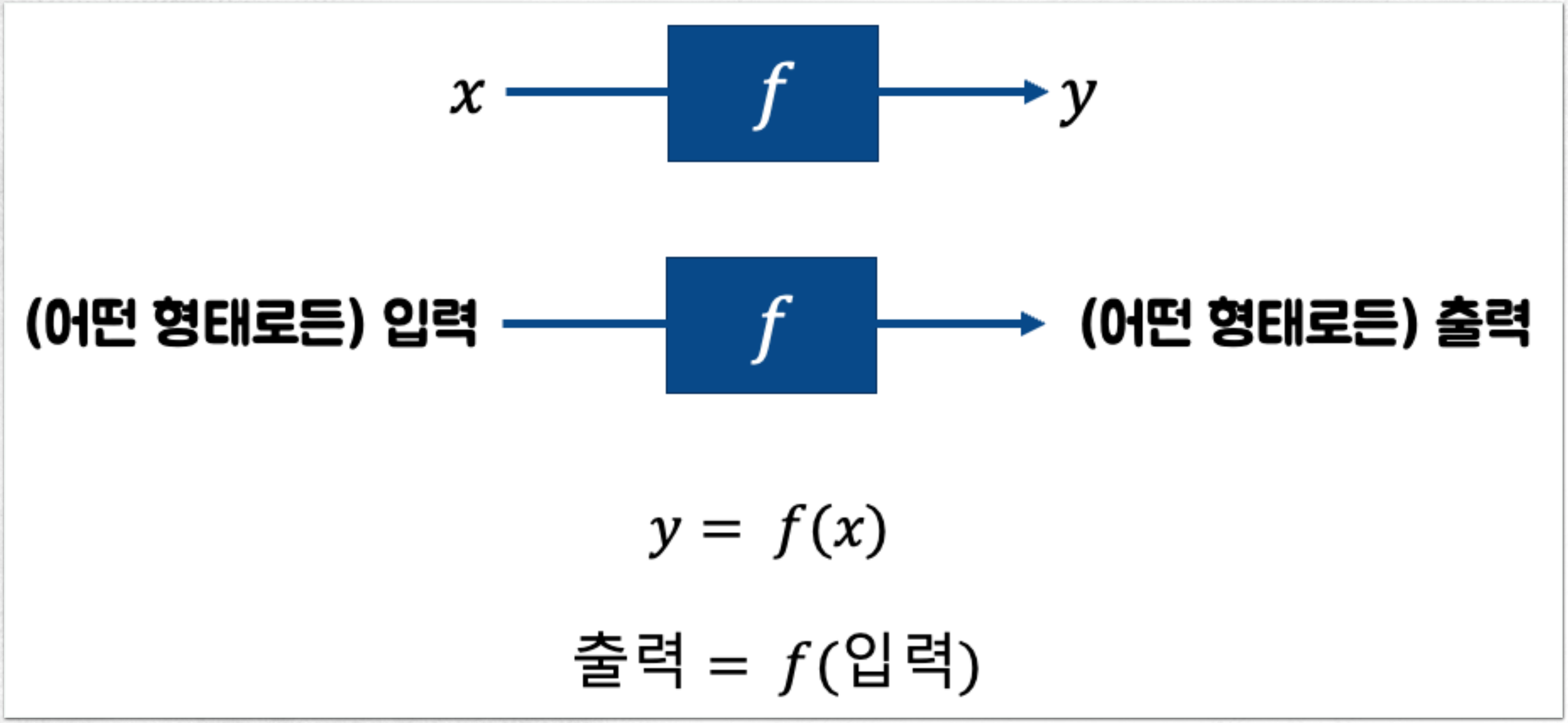
$$z = f(x, y) = x^2y + 4x + 1$$

$$f(2,3) = 21$$

- Python build-in functions

```
>>> abs(-9)
9
>>> pow(3,2)
9
>>> round(4.3)
4
>>> pow( abs(-2), round(4.3) )
16
>>> round(-3.5)
-4
```

Functions



$$y = f(x)$$

$$\text{출력} = f(\text{입력})$$

Typecast

- Functions that convert from one type to another

```
>>> int(34.6)
34
>>> int(-4.3)
-4
>>>float(21)
21.0
```

help( )

```
>>> help(abs)
Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.

>>> help(pow)
Help on built-in function pow in module builtins:

pow(x, y, z=None, /)
    Equivalent to x**y (with two arguments) or x**y % z (with three arguments)

    Some types, such as ints, are able to use a more efficie
    invoked using the three argument form.

>>> help(round)
Help on built-in function round in module builtins:

round(...)
    round(number[, ndigits]) -> number

    Round a number to a given precision in decimal digits (d
    This returns an int when called with one argument, other
    same type as the number.  ndigits may be negative.

>>>
```

```
>>> pow(2,4)
16
>>> pow(2,4,3)
1
>>> round(3.141592)
3
>>> round(3.141592,2)
3.14
```

id( ) : Memory addresses

```
>>> help(id)
Help on built-in function id in module builtins:

id(obj, /)
    Return the identity of an object.

    This is guaranteed to be unique among simultaneously existing objects.
    (CPython uses the object's memory address.)
```


```
>>> id(-9)
2112378679184
>>> id(23.1)
2112332606104
>>> shoe_size = 8.5
>>> id(shoe_size)
2112332606080
>>> shoe_size = 9.0
>>> id(shoe_size)
2112332606104
>>> id(abs)
2112332504736
>>> id(round)
2112332531824
>>>
```

Defining your own functions

```
>>> convert_to_celsius(212)
100.0
```

```
>>> convert_to_celsius(212)
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    convert_to_celsius(212)
NameError: name 'convert_to_celsius' is not defined
```

```
>>> def convert_to_celsius (fahrenheit):
        return (fahrenheit - 32) * 5/9
```

How Python executes

```
>>> def convert_to_celsius (fahrenheit):
        return (fahrenheit - 32) * 5/9

>>> convert_to_celsius(212)
100.0
>>>
```

How Python executes

```
>>> def convert_to_celsius (fahrenheit):
        return (fahrenheit - 32) * 5/9
```

- Python executes the function definition – creates the function object

How Python executes

```
>>> def convert_to_celsius (fahrenheit):
        return (fahrenheit - 32) * 5/9

>>> convert_to_celsius(212)
```

- Python executes the function definition – creates the function object
- Python executes function call
  - It assigns 212 to fahrenheit

How Python executes

```
>>> def convert_to_celsius (fahrenheit):
        return (fahrenheit - 32) * 5/9


>>> convert_to_celsius(212)
100.0
```

- Python executes the function definition – creates the function object
- Python executes function call
  - It assigns 212 to fahrenheit
- Python executes the return statement
  - (212 – 32) * 5/9
  - The result of calling convert_to_celsius(212) is 100.0

How Python executes

```
>>> def convert_to_celsius (fahrenheit):
        return (fahrenheit - 32) * 5/9


>>> convert_to_celsius(212)
100.0
>>>
```

- Python executes the function definition – creates the function object
- Python executes function call
  - It assigns 212 to fahrenheit
- Python executes the return statement
  - (212 – 32) * 5/9
  - The result of calling convert_to_celsius(212) is 100.0
- Once Python has finished executing the function call, it returns to the place where the function was originally called.

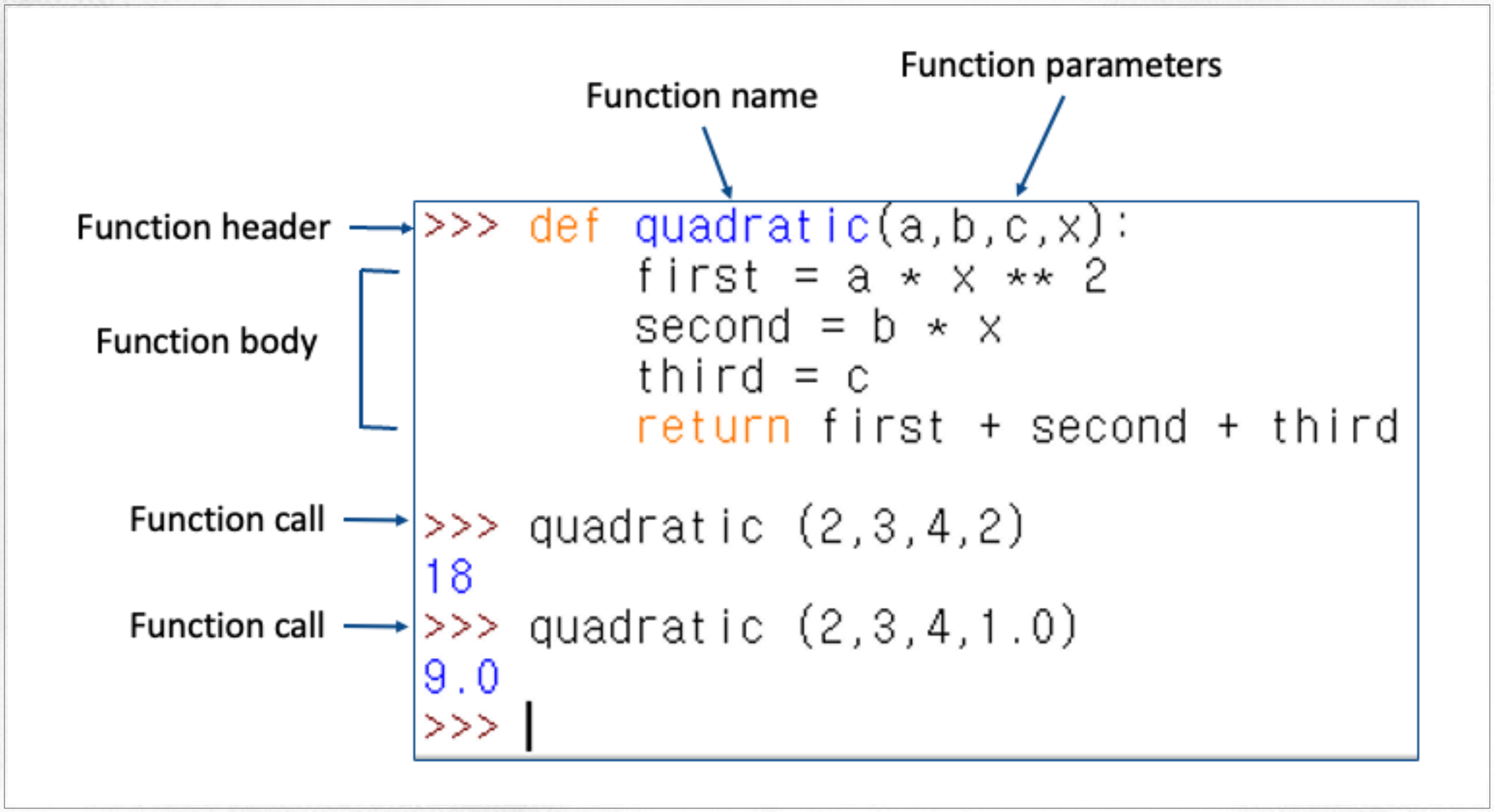
Keywords

- We cannot use `def` and `return` as either variable names or as function names

```
>>> def = 3
SyntaxError: invalid syntax
```

Local variables

- Local variables are created within a function
- first, second, third are local variables of the function quadratic
- Function parameters are also local variables



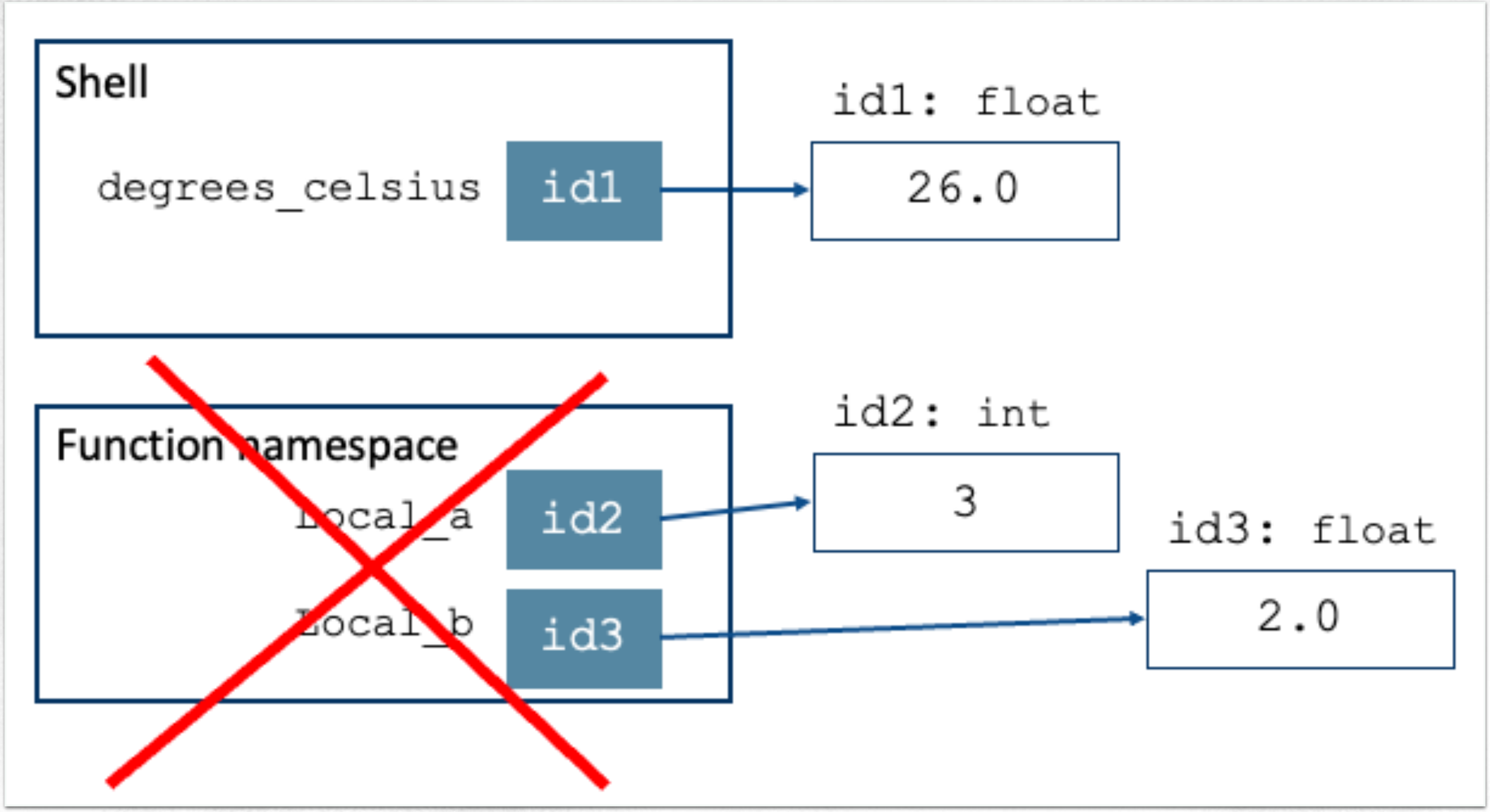
Errors

- Number of parameters
- Redefinition is ok
- Local variables

```
>>> def quadratic(a,b,c,x):
        first = a * X ** 2
        second = b * x
        third = c
        return first + second + third

>>> quadratic (2,3,4,2)
18
>>> quadratic (2,3,4,1.0)
9.0
>>> quadratic ( 2,3,4)
Traceback (most recent call last):
  File "<pyshell#68>", line 1, in <module>
    quadratic ( 2,3,4)
TypeError: quadratic() missing 1 required positional argument: 'x'
>>> def quadratic (a,b,x):
        first = a * x **2
        second = b * x
        return first + second

>>> quadratic ( 2,3,4,2)
Traceback (most recent call last):
  File "<pyshell#75>", line 1, in <module>
    quadratic ( 2,3,4,2)
TypeError: quadratic() takes 3 positional arguments but 4 were given
>>> quadratic(2,3,2)
14
>>> first
Traceback (most recent call last):
  File "<pyshell#77>", line 1, in <module>
    first
NameError: name 'first' is not defined
```

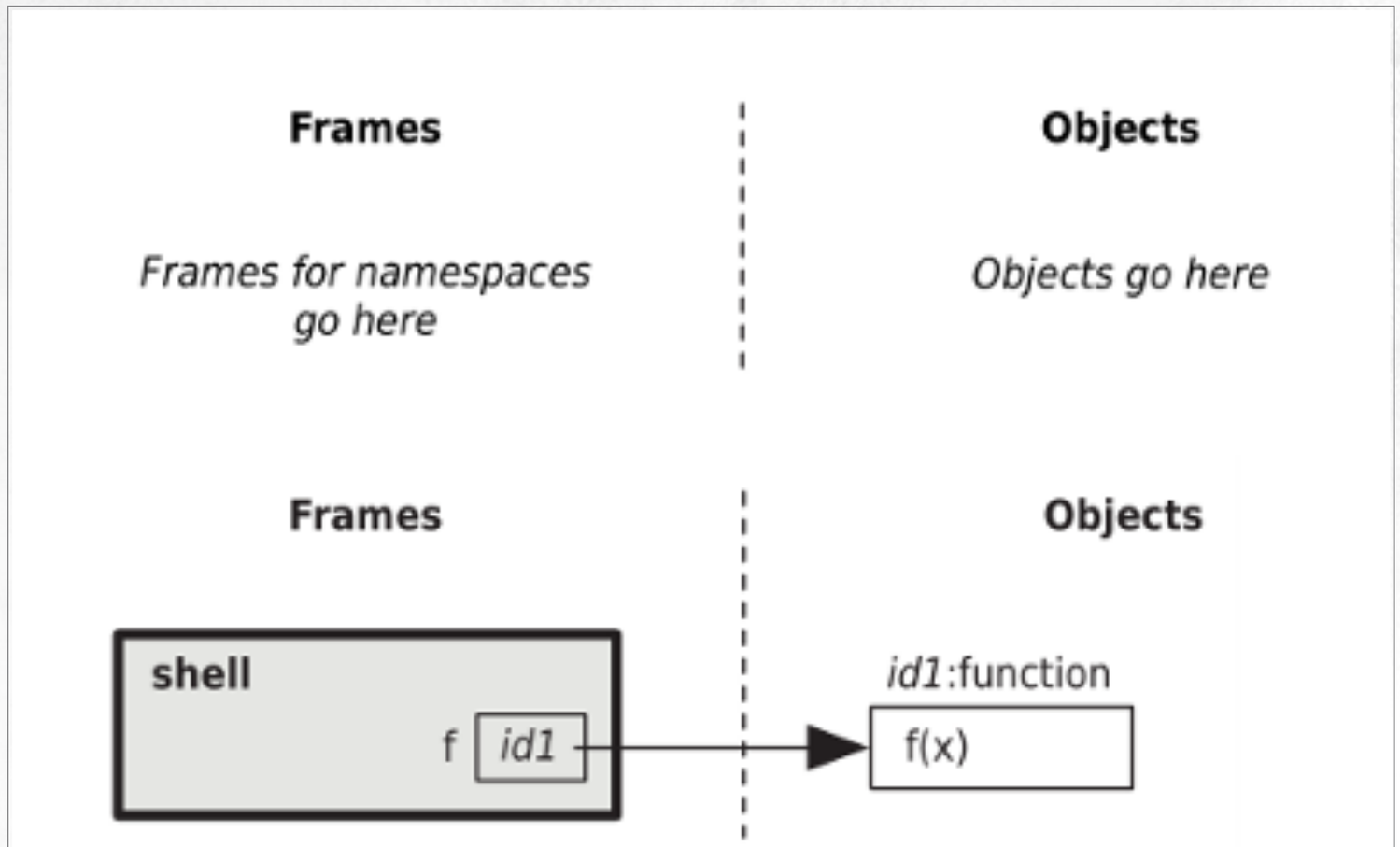
Local variables and namespaces

- When Python executes a function call, it creates a namespace to store local variables for that call
- When the function returns, the namespace is no longer tracked.



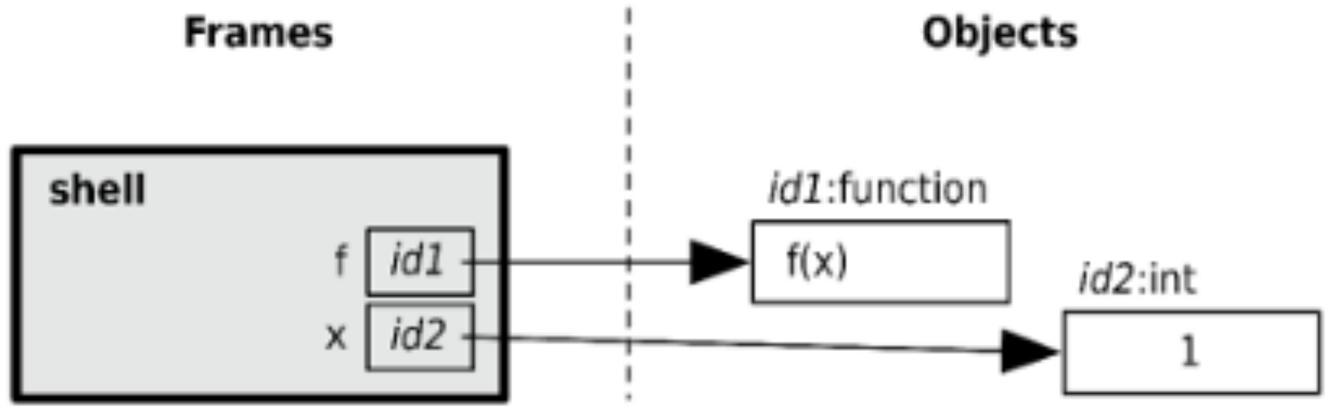
Local variables and namespaces

```
>>> def f(x):
        x = 2 * x
        return x

>>> x = 1
>>> x = f(x + 1) + f(x + 2)
>>> x
```



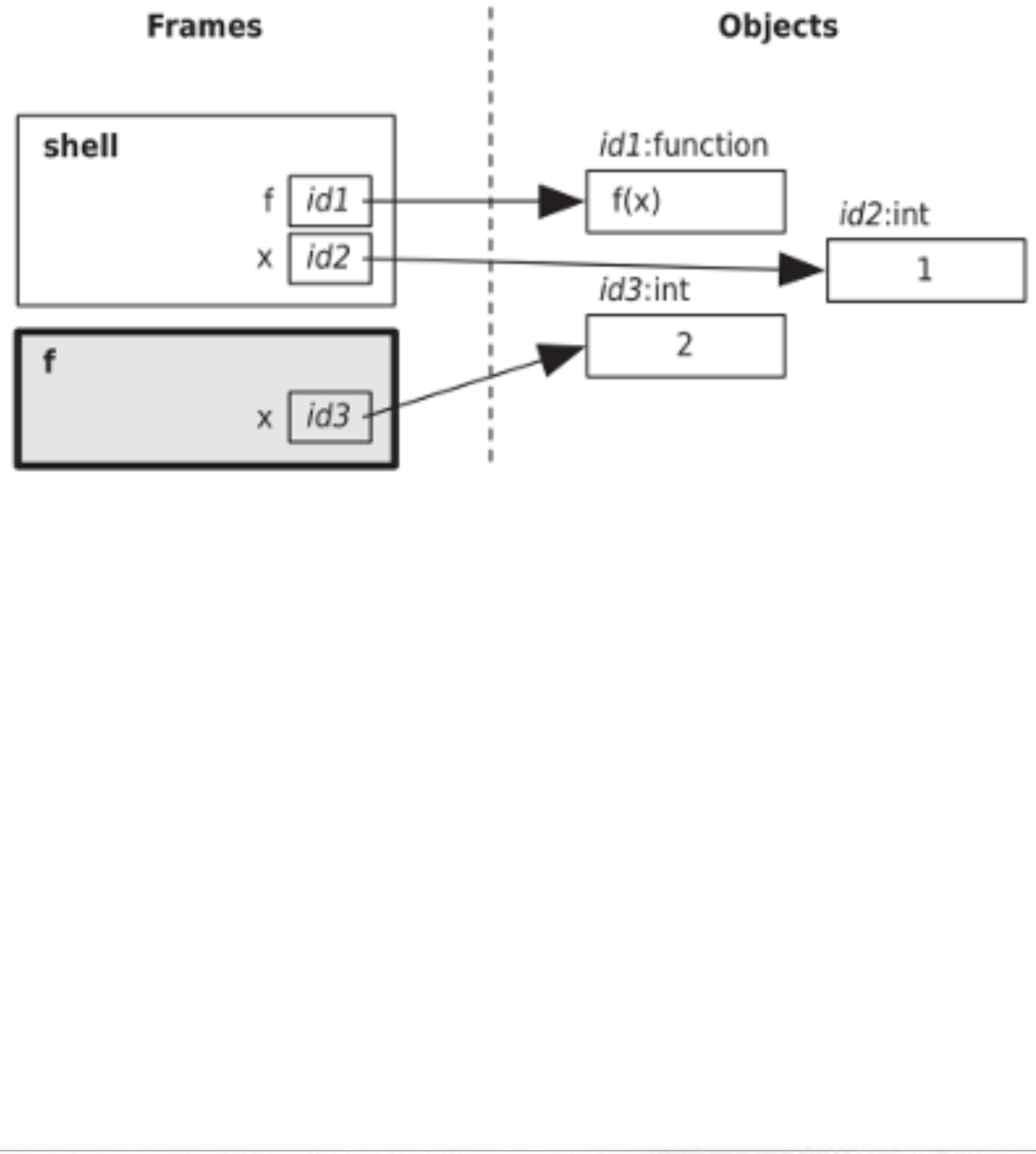
Local variables and namespaces

```
>>> def f(x):
        x = 2 * x
        return x

>>> x = 1
>>> x = f(x + 1) + f(x + 2)
>>> x
```



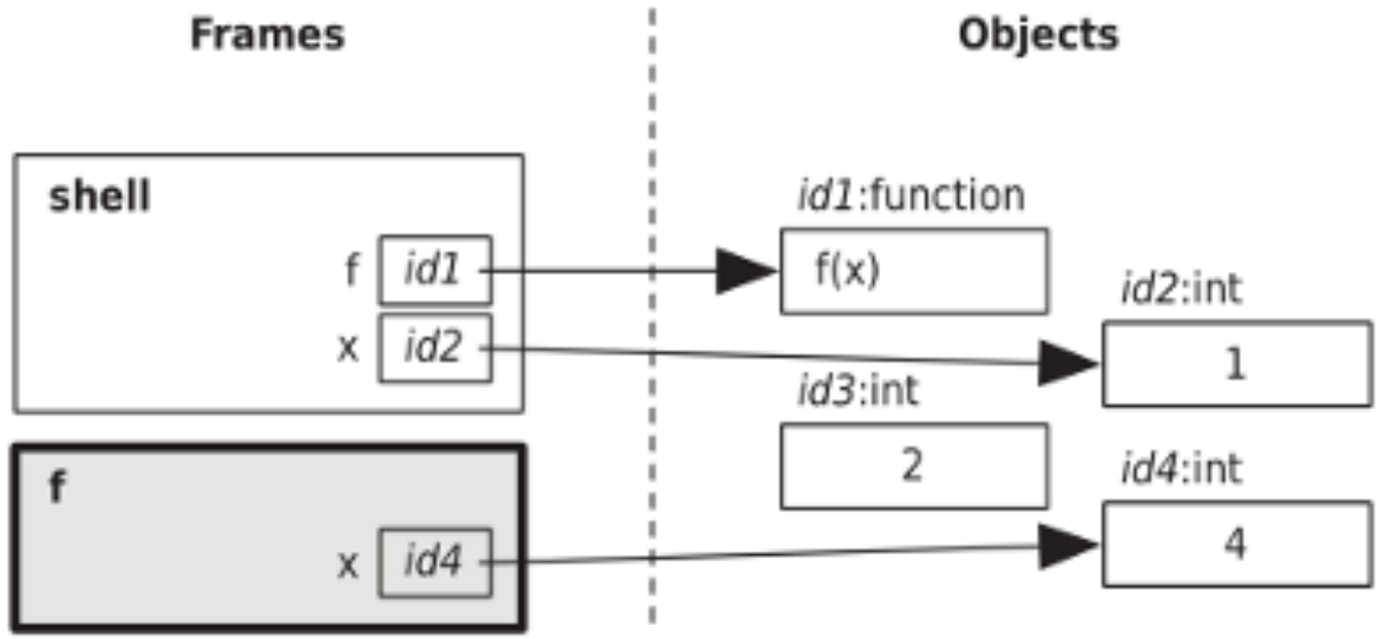
Local variables and namespaces

```
>>> def f(x):
        x = 2 * x
        return x

>>> x = 1
>>> x = f(x + 1) + f(x + 2)
>>> x
```



Local variables and namespaces

```
>>> def f(x):
        x = 2 * x
        return x

>>> x = 1
>>> x = f(x + 1) + f(x + 2)
>>> x
```

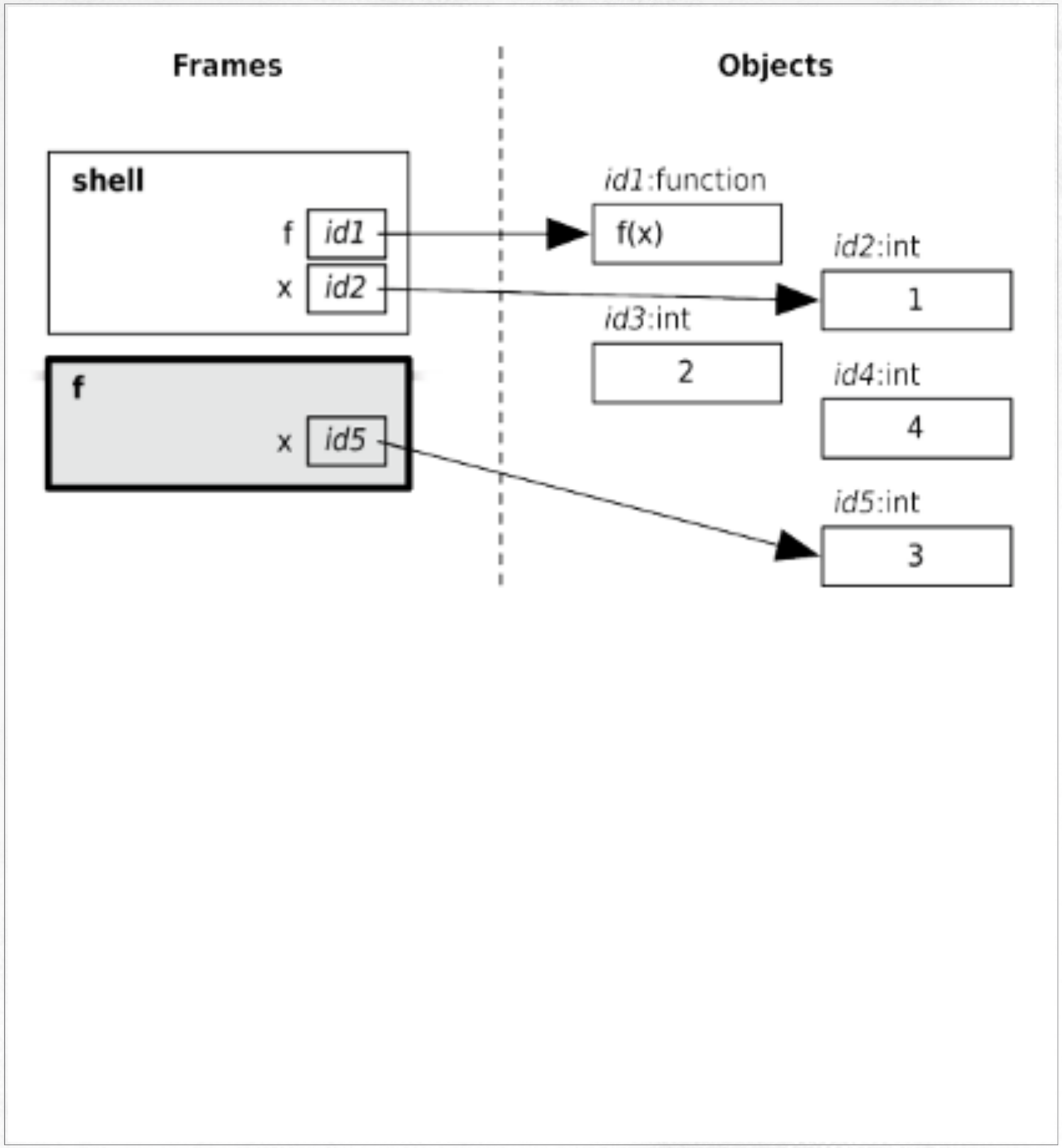


Local variables and namespaces

```
>>> def f(x):
        x = 2 * x
        return x

>>> x = 1
>>> x = f(x + 1) + f(x + 2)
>>> x
```

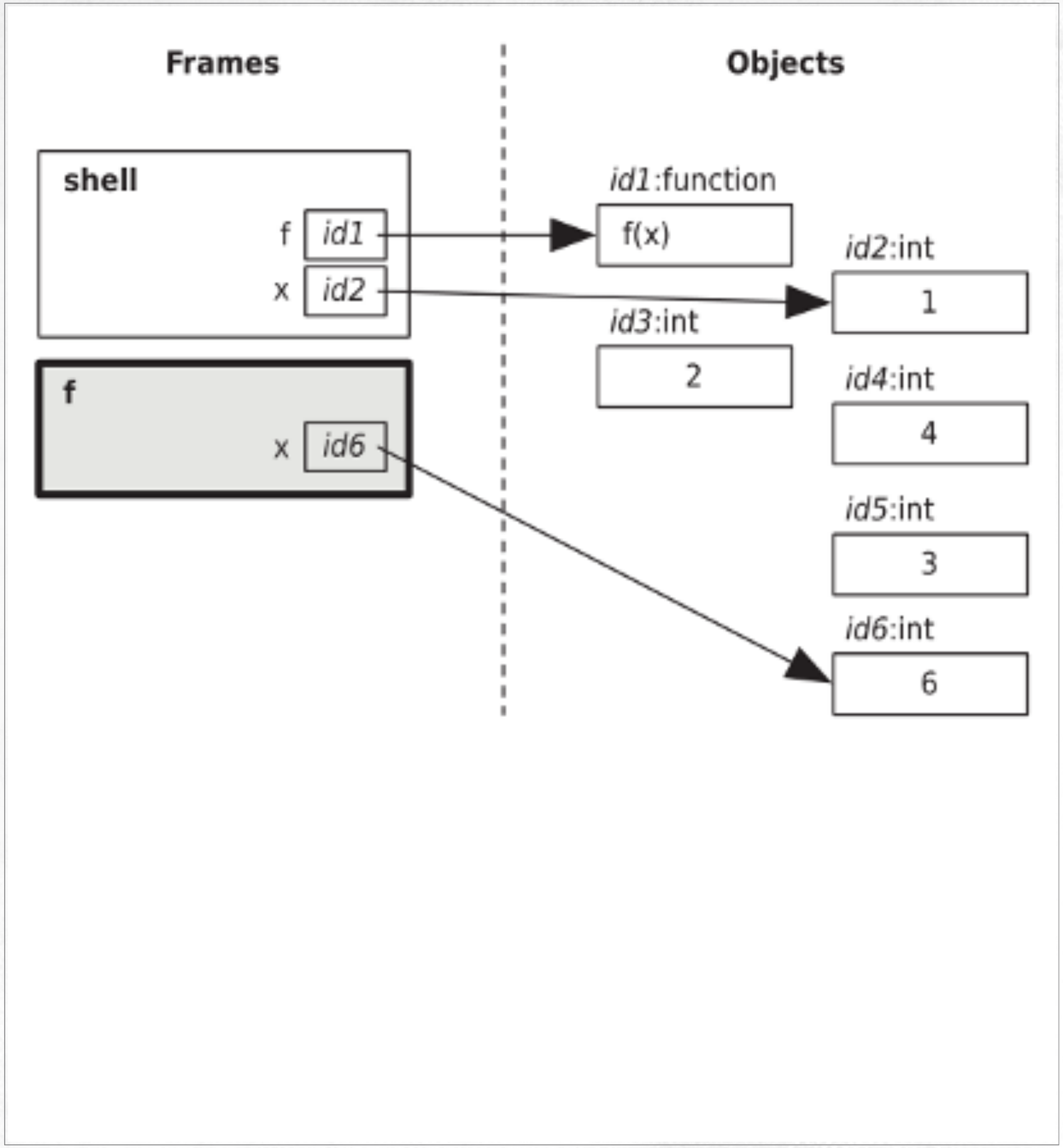
Local variables and namespaces

```
>>> def f(x):
        x = 2 * x
        return x

>>> x = 1
>>> x = f(x + 1) + f(x + 2)
>>> x
```



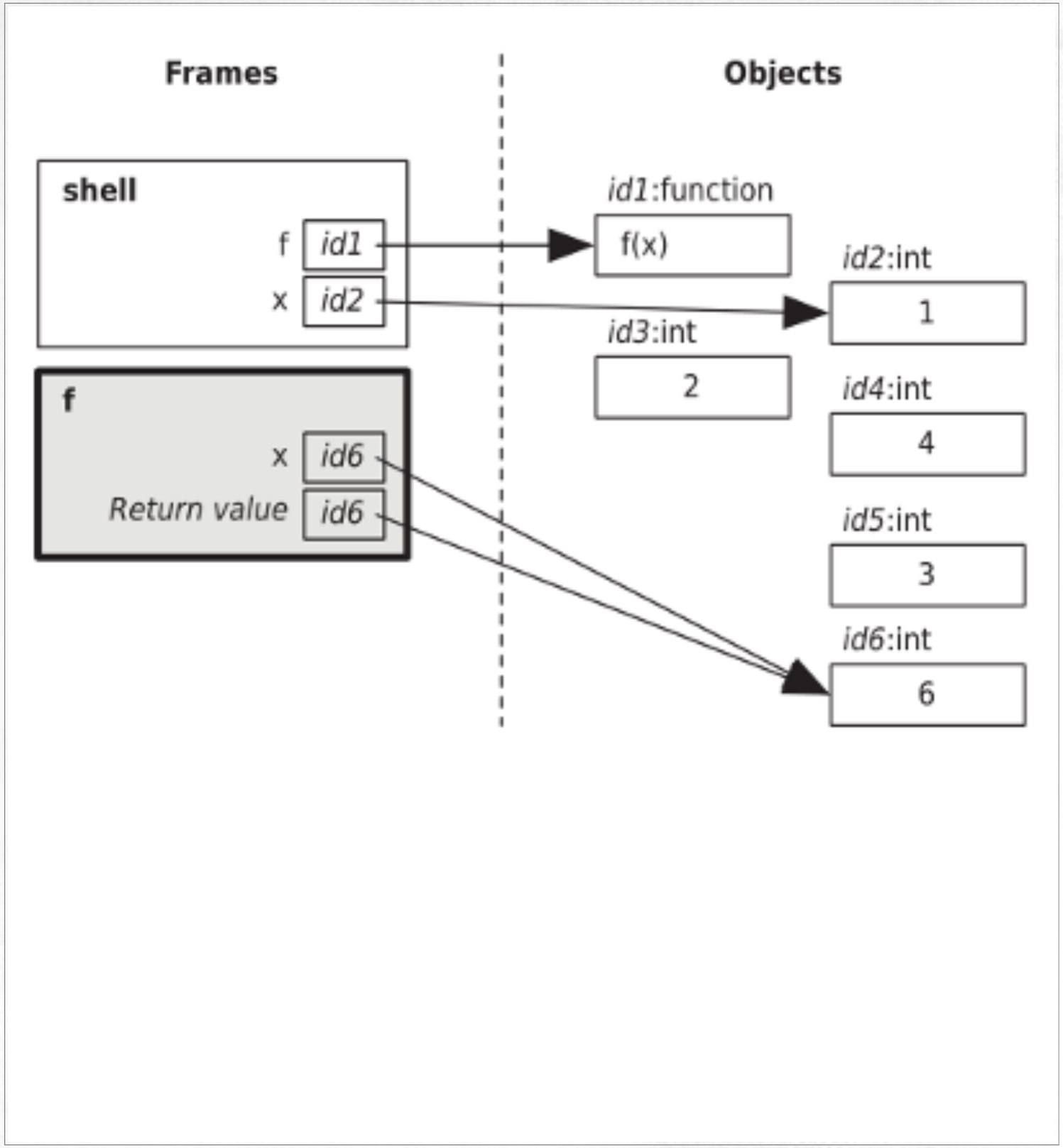
Local variables and namespaces

```
>>> def f(x):
        x = 2 * x
        return x

>>> x = 1
>>> x = f(x + 1) + f(x + 2)
>>> x
```



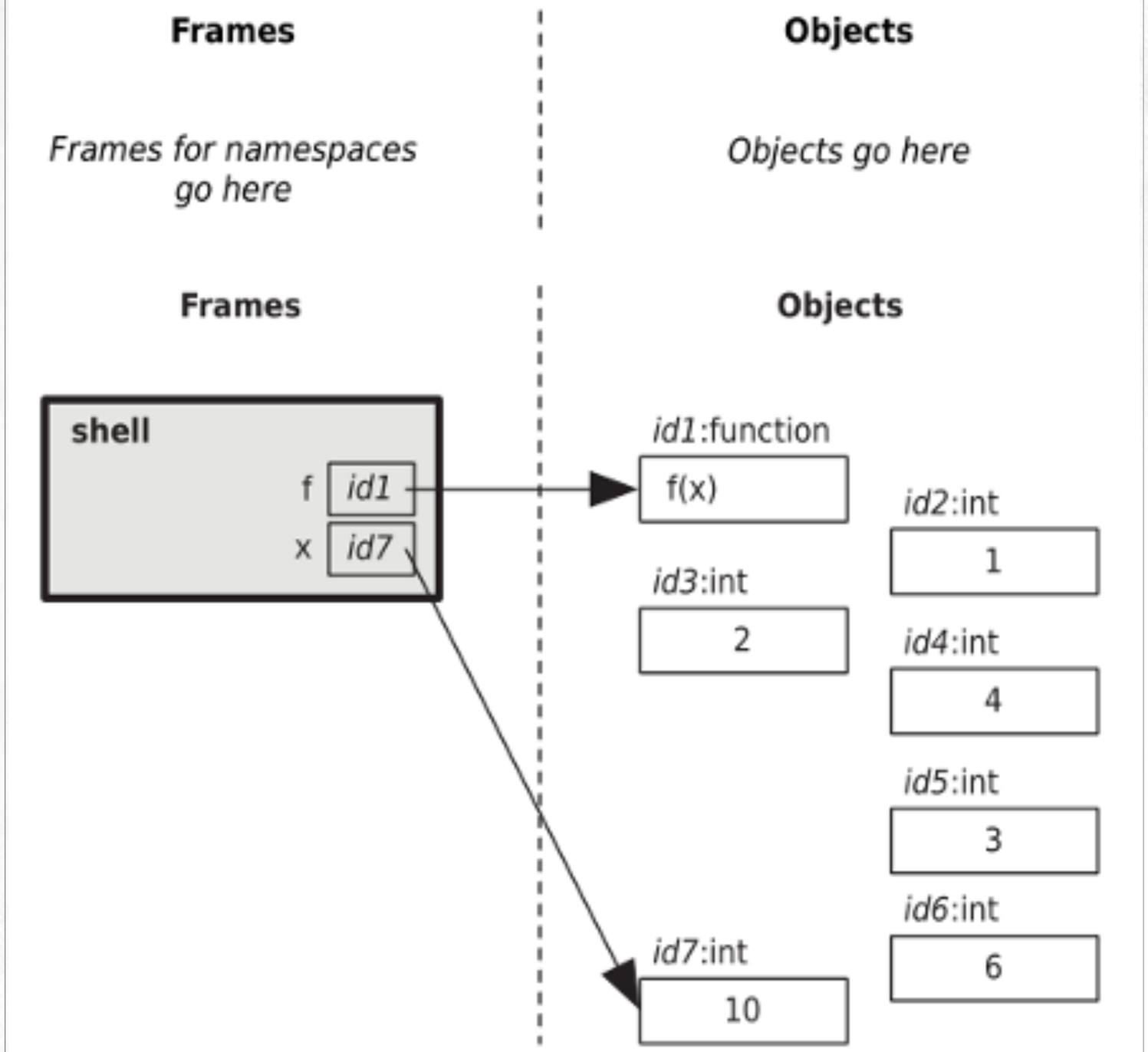
Local variables and namespaces

```
>>> def f(x):
        x = 2 * x
        return x

>>> x = 1
>>> x = f(x + 1) + f(x + 2)
>>> x
```



Local variables and namespaces

```
>>> def f(x):
        x = 2 * x
        return x

>>> x = 1
>>> x = f(x + 1) + f(x + 2)
>>> x
```



Better understanding of return value

- To better understand the return value of the function

Editor

```
def sum(a,b):
    return a+b


sum(3,4)
```

Shell

```
RESTART:
C:/Users/jiyoung/AppD
ata/Local/Programs/Py
thon/Python35/Scripts
/test01.py
```

Better understanding of return value

- To better understand the return value of the function

Editor

```
def sum(a,b):
    return a+b

result = sum(3,4)
```

Shell

```
RESTART:
C:/Users/jiyoung/AppD
ata/Local/Programs/Py
thon/Python35/Scripts
/test01.py
```

Better understanding of return value

- To better understand the return value of the function

Editor

```
def sum(a,b):
    return a+b

result = sum(3,4)
print(result)
```

Shell

```
RESTART:
C:/Users/jiyoung/AppD
ata/Local/Programs/Py
thon/Python35/Scripts
/test01.py

7
```

Better understanding of return value

- To better understand the return value of the function

Editor

```
def sum(a,b):
    return a+b


print(sum(3,4))
```

Shell

```
RESTART:
C:/Users/jiyoung/AppD
ata/Local/Programs/Py
thon/Python35/Scripts
/test01.py

7
```

Function with no parameters

- Function with no input
- Just execution
- return value is **'Hello'** and print nothing

```
def say():
    return 'Hello'

say()
```

Function with no parameters

- Function with no input
- Just execution
- return value is **'Hello'** and print **'Hello'**

```
def say():
    return 'Hello'

print(say())
```

Function with no parameters

- Function with no input
- Just execution
- return value is **'Hello'** and print **'Hello'**

```
def say():
    return 'Hello'

word = say()
print(word)
```

Function with no parameters

- Every function has only one return value
- If the return value is not defined in the function definition, the return value is **None**

```
def say():
    print('Hello')

say()
```

Function with no parameters

- Every function has only one return value
- If the return value is not defined in the function definition, the return value is **None**

```
def say():
    print('Hello')

print(say())
```

Function with no parameters

- Every function has only one return value
- If the return value is not defined in the function definition, the return value is **None**

```
def say():
    print('Hello')

word = say()
print(word)
```

Designing a new function

- Writing a good essay
  - A topic
  - Background material
  - An outline
  - Filling in the outline with details
- Writing a good function
  - An idea
  - A name
  - Parameters
  - A return value
  - Function body (the details)

Docstring

- Documentation string
- For humans to read
  - For yourself
  - For co-workers
  - For sharing

```
def days_difference(day1, day2):
    """ (int, int) -> int

    Return the number of days between day1 and day2,
    which are both in the range 1-365
    (thus indicating the day of the year).


    >>> days_difference(200, 224)
    24
    >>> days_difference(50, 50)
    0
    >>> days_difference(100, 99)
    -1
    """
    return day2 - day1
```

Docstring

Function header → `def days_difference(day1, day2):`

Start of the docstring → `"""` `(int, int) -> int`

1) Types of (Parameters) -> Return value

2) Function description →

```
Return the number of days between day1 and day2,
which are both in the range 1-365
(thus indicating the day of the year).
```

3) Example calls →

```
>>> days_difference(200, 224)
24
>>> days_difference(50, 50)
0
>>> days_difference(100, 99)
-1
```

End of the docstring → `"""`

Function body →

```
return day2 - day1
```

Function design recipe

- Examples
  - What arguments/parameters to give
  - What information it will return
  - Pick a function name
- Type contract
  - Types of parameters and return value
- Header
  - Give parameters names
- Description
- Body
- Test

```
def days_difference(day1, day2):
    """ (int, int) -> int


    Return the number of days between day1 and day2,
    which are both in the range 1-365
    (thus indicating the day of the year).


    >>> days_difference(200, 224)
    24
    >>> days_difference(50, 50)
    0
    >>> days_difference(100, 99)
    -1
    """
    return day2 - day1
```

WARNING

- Read: https://docs.python.org/3.7/tutorial/floatingpoint.html
- Floating-point numbers are represented in computer hardware as base 2 (binary) fractions.
- Decimal fraction vs. binary fraction

```
>>> 0.1+0.2
0.30000000000000004
>>> round(4.5)
4
>>>
```

Summary

- Mathematicians create functions to make calculations (such as Fahrenheitto-Celsius conversions) easy to reuse and to make other calculations easier to read because they can use those functions instead of repeatedly writing out equations.
- Programmers do this too, at least as often as mathematicians.
- In this chapter we explored several of the built-in functions that come with Python, and we also learned how to define programmer's own functions.

Thank you

경희대학교
KYUNG HEE UNIVERSITY