CS217: Computer Programming Language: Functions

Jakramate Bootkrajang
Based on material by Kittipitch Kuptavanich for 204217 15s1

Outline

- What is function ?
- Why use function ?
- Built-in function and User-defined function
- Printing related functions
- Keyword argument
- Exercise

What is a Function ?

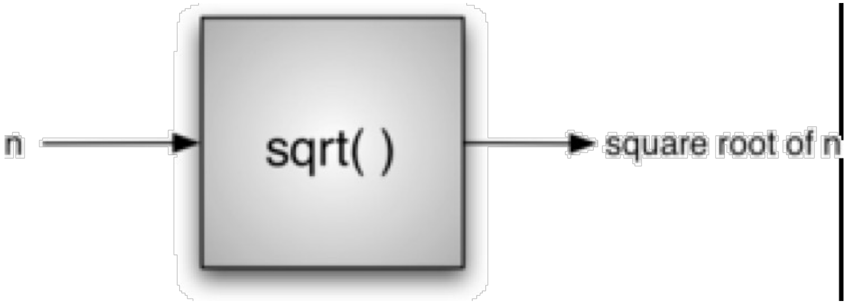
- A named set of codes for some specific purpose

```
>>> type(32)
<class 'int'>
```

- Here, the function name is `type’
- 32 is called function argument (parameter)
- The function returns a result, in which case is the type of the numeric data, 32.

Abstraction with function

- Function abstracts `the process for getting the desired output' (blackboxing)
- We do not need to know how sqrt() figures out the answer



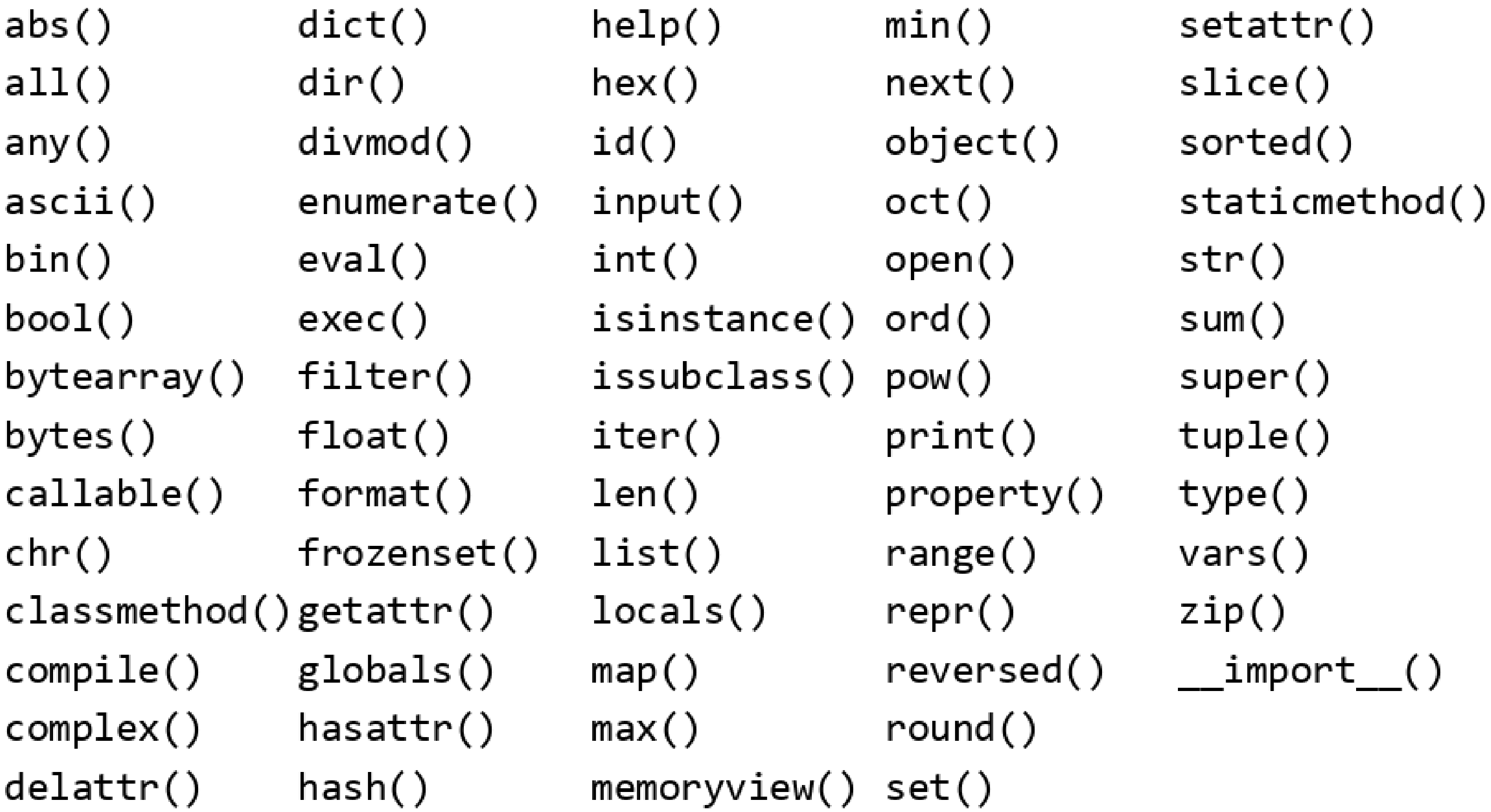
- We only need to know
  - Function name and list of its arguments
  - what does it return ?

Why using function ?

- The code will be more readable
- The code will be easier to maintain
- The code will be reusable
- Follow divide-and-conquer problem solving methodology
  - Function can contains functions which also contain functions... and so on.

Built-in functions

- Functions that come with Python distributions or packages



abs()
all()
any()
ascii()
bin()
bool()
bytearray()
bytes()
callable()
chr()
classmethod()
compile()
complex()
delattr()
dict()
dir()
divmod()
enumerate()
eval()
exec()
filter()
float()
format()
frozenset()
getattr()
globals()
hasattr()
hash()
help()
hex()
id()
input()
int()
isinstance()
issubclass()
iter()
len()
list()
locals()
map()
max()
memoryview()
min()
next()
object()
oct()
open()
ord()
pow()
print()
property()
range()
repr()
reversed()
round()
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()
tuple()
type()
vars()
zip()
__import__()

User-defined function

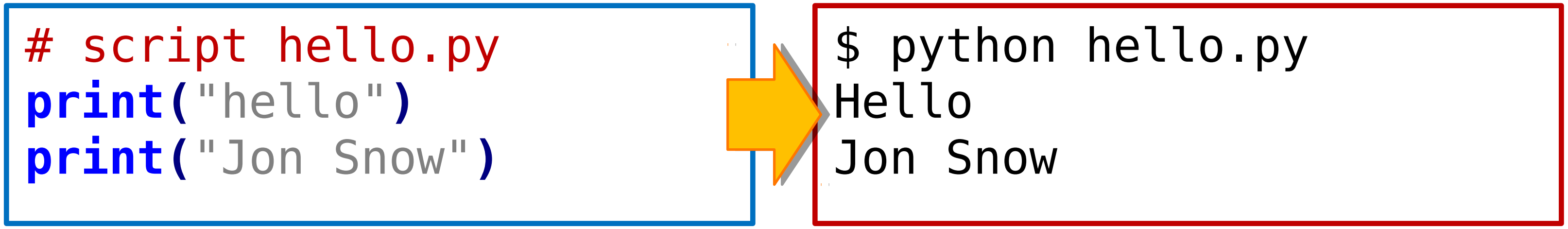
- A function defined specifically for a task by user
- Use keyword def to define a function in one of the following form
  - def my_function()
  - def my_function(param1, param2)
  - def my_function(param1=10)
- Return the result computed in the function using keyword return

Important functions/methods

- Functions involving information display
  - The print() function
  - The format() method for string object
- Note
  - A function is a set of codes which works for many data types
  - A method is a function which is applied on some specific data object only

The print() function

- For printing information on the screen
- General usage
  - print(string)
- By default print() adds newline character (‘\n’) at the end of the line

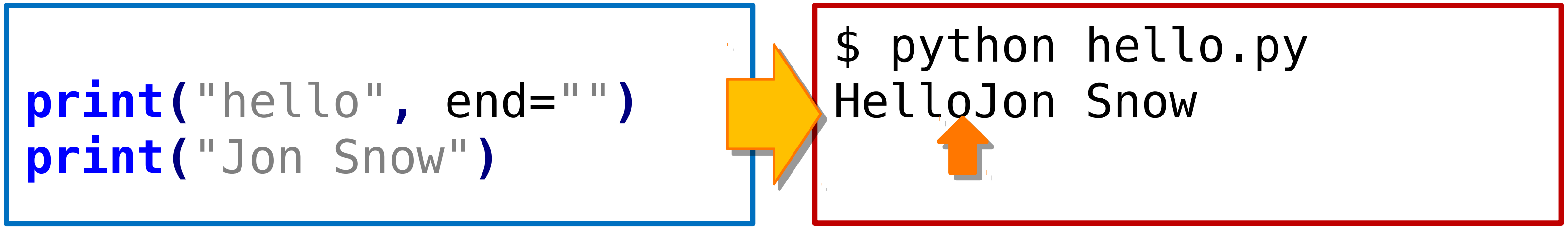


```
# script hello.py
print("hello")
print("Jon Snow")
```

```
$ python hello.py
Hello
Jon Snow
```

Optional parameter

- We can omit (‘\n’) by specifying keyword argument end=””



```python
print("hello", end="")
print("Jon Snow")
```

```
$ python hello.py
HelloJon Snow
```

- “end” is an example of optional parameter
- Optional parameter is not required when a function is called
  - When not specified, optional parameter takes its default value
  - The default value for “end” is ‘\n’

The print() function [2]

- Different kind of characters can be passing as “end” for example

```
print("hello", end="**")
print("Jon Snow")
```

```
$ python hello.py
Hello**Jon Snow
```

- There is also “sep” optional parameter for separating the input parameters

```
# script number.py
print(1, 2, 3)
print(1, 2, 3, sep="")
print(1, 2, 3, sep="**")
```

```
$ python numbers.py
1 2 3
123
1**2**3
```

Special Characters

Escape Sequence	Meaning	Notes
\\	Backslash (\)	
\'	Single quote (')	
\"	Double quote (")	
\a	ASCII Bell (BEL)	
\b	ASCII Backspace (BS)	
\f	ASCII Formfeed (FF)	
\n	ASCII Linefeed (LF)	
\r	ASCII Carriage Return (CR)	
\t	ASCII Horizontal Tab (TAB)	
\v	ASCII Vertical Tab (VT)	
\ooo	Character with octal value *ooo*	
\xhh	Character with hex value *hh*	

The format() method

- Alternatively, we can format the string before passing it to print() function using the method str.format()

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

- The number in parentheses is a placeholder for placing respective parameters. It starts from 0

Side note: Function call vs Method call

- Calling a function we simply write
  - function_name(parameter)
- Calling a method of some data object, we write
  - object.method_name(parameter)
  - For example
    - string.isdigit() # if string is all digits

The format() method [2]

- It is possible to display number in many fancy ways by using “:” (colon)

```
>>> print('PI is approximately
{0:.3f}.'.format(math.pi))
PI is approximately 3.142.
```

```
>>> print('PI is approximately
{0:09.3f}.'.format(math.pi))
PI is approximately 00003.142.
```

- .3f indicates 3 decimal points
- 09.3f indicates total digits, filling left most with 0 if there’re not enough digit to display
- Using #9 instead of replaces 0 with space

More format() examples

- Aligning text and specifying a width

```
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'           centered           '
# use '*' as a fill char
>>> '{:*^30}'.format('centered')
'***********centered***********'
```

More format() examples [2]

```
# specify decimal point
>>> '{:5.2f}; {:5.3f}'.format(3.14, -3.14)
' 3.14; -3.140'
# show it always
>>> '{:+f}; {:+f}'.format(3.14, -3.14)
'+3.140000; -3.140000'
# show a space for positive numbers
>>> '{: f}; {: f}'.format(3.14, -3.14)
' 3.140000; -3.140000'
# show only the minus -- same as '{:f}; {:f}'
>>> '{:-f}; {:-f}'.format(3.14, -3.14)
'3.140000; -3.140000'
```

More format() examples [3]

- Using comma as a thousands separator:

```
>>> '{:,}'.format(1234567890)
'1,234,567,890'
```

- Expressing percentage

```
>>> points = 19
>>> total = 22
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 86.36%'
```

- Using type-specific formatting

```
>>> import datetime
>>> d = datetime.datetime(2015, 7, 4, 12, 15, 58)
>>> '{:%Y-%m-%d %H:%M:%S}'.format(d)
'2015-07-04 12:15:58'
```

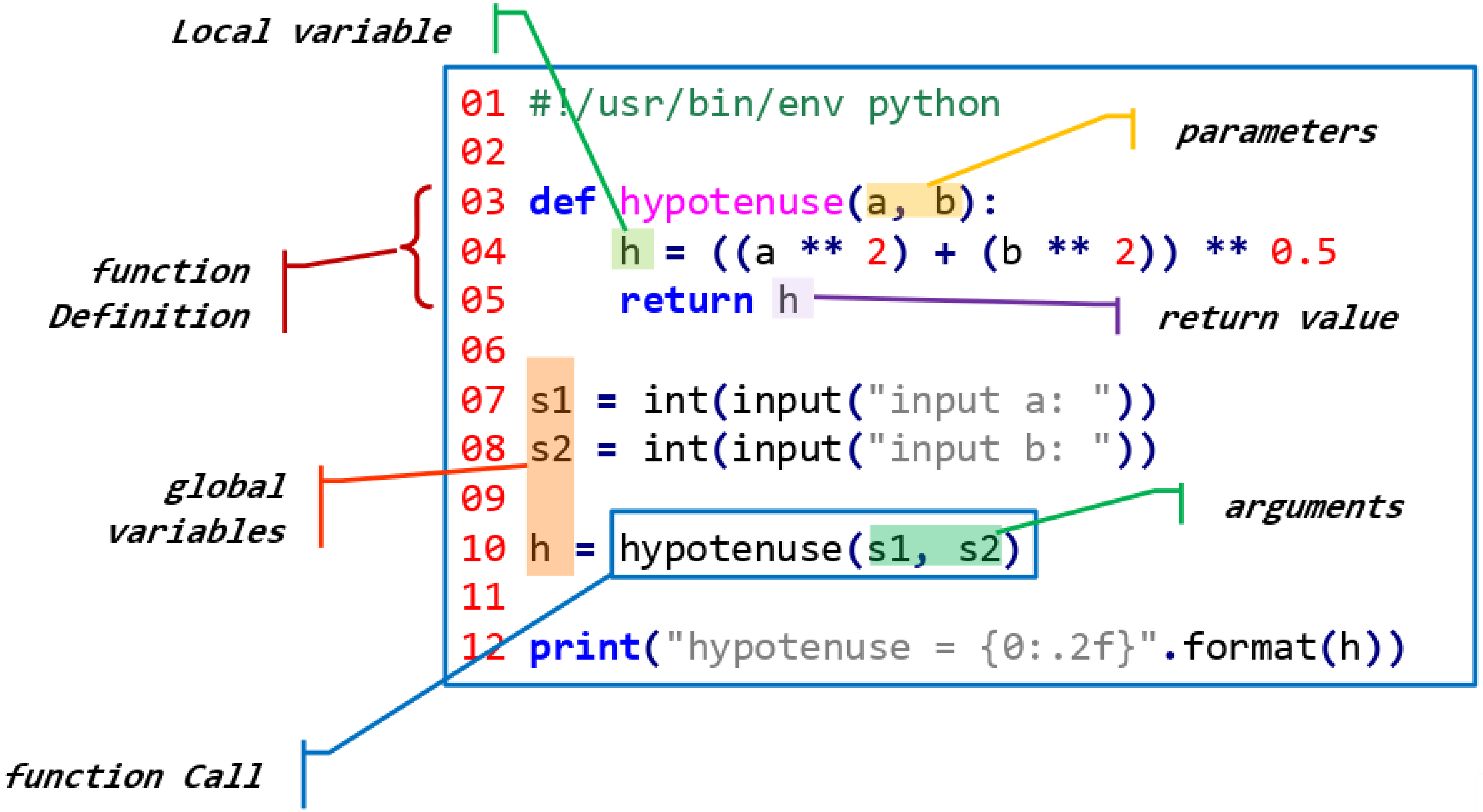
Void and Fruitful function

- There are 2 types of functions
  - Function that returns something (fruitful function)
    - For example math.abs()
  - Function that returns nothing (void function)
    - For example print()

Working with fruitful function

- Two ways of using the result returned by a function
  - Declare a variable for collecting value from a fruitful function
  - Directly call the function within an expression

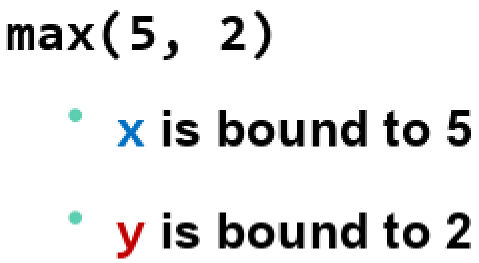
Anatomy of Python function



```python
#!/usr/bin/env python

def hypotenuse(a, b):
    h = ((a ** 2) + (b ** 2)) ** 0.5
    return h

s1 = int(input("input a: "))
s2 = int(input("input b: "))

h = hypotenuse(s1, s2)

print("hypotenuse = {0:.2f}".format(h))
```

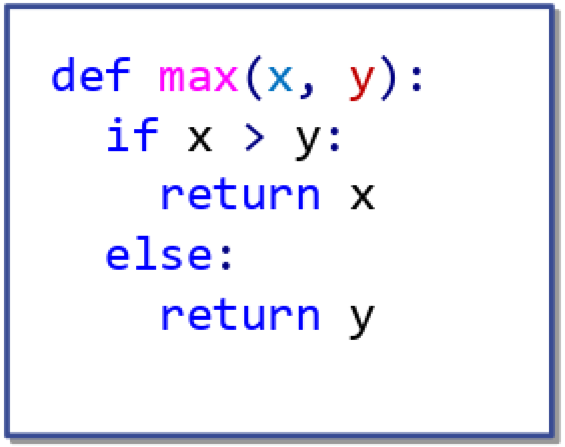
Keyword arguments and default value

- In python there are two ways for formal parameters to get bound to actual parameters
- Positional
  - First formal parameter is bound to the first actual parameter, the 2nd formal to the 2nd actual, and so on



```
max(5, 2)
```

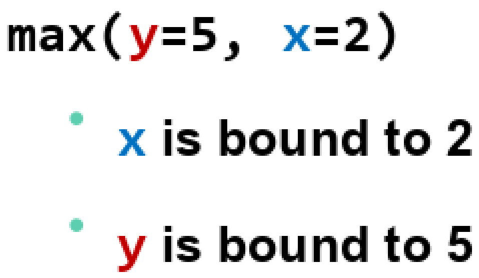
- x is bound to 5
- y is bound to 2



```
def max(x, y):
  if x > y:
    return x
  else:
    return y
```

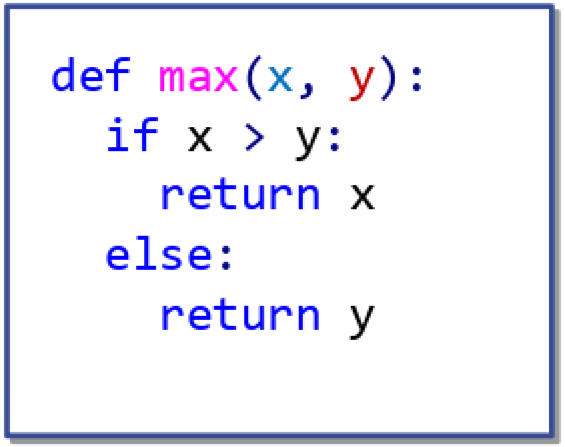
Keyword arguments and default value [2]

- Keyword arguments
  - Binding using the name of the formal parameters



```
max(y=5, x=2)
```

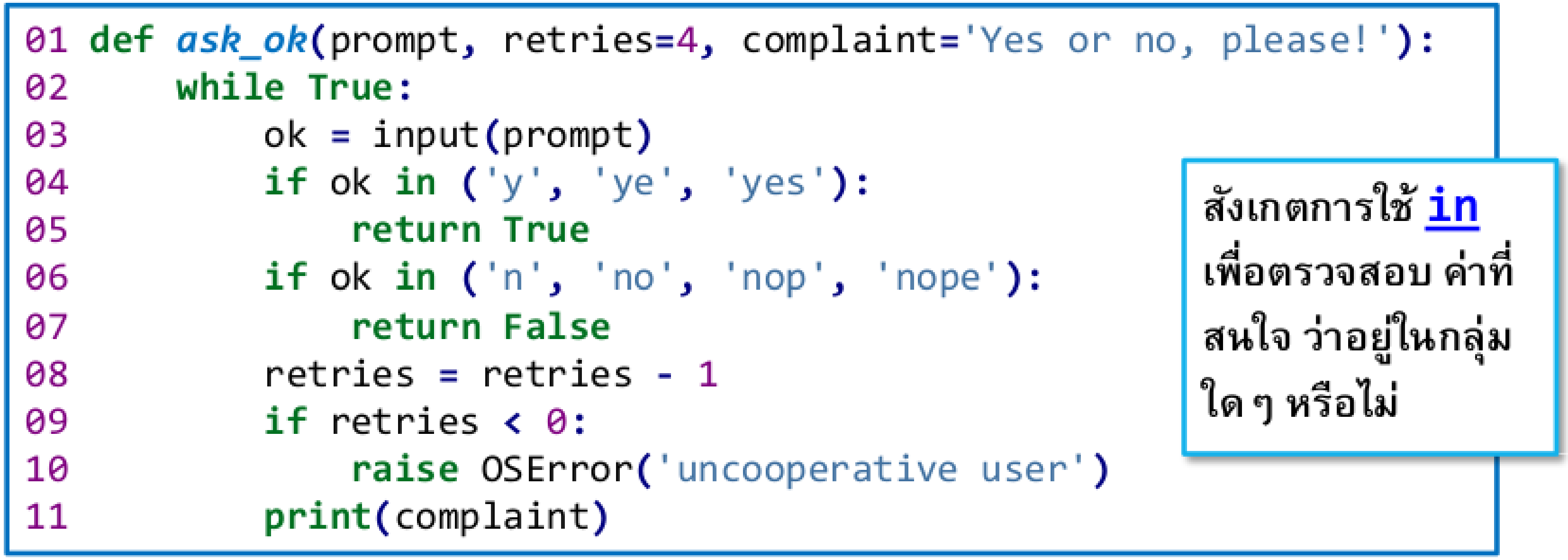
- x is bound to 2
- y is bound to 5



```
def max(x, y):
  if x > y:
    return x
  else:
    return y
```

Keyword arguments and default value [3]

- The most useful form is to specifying a default value for one or more arguments

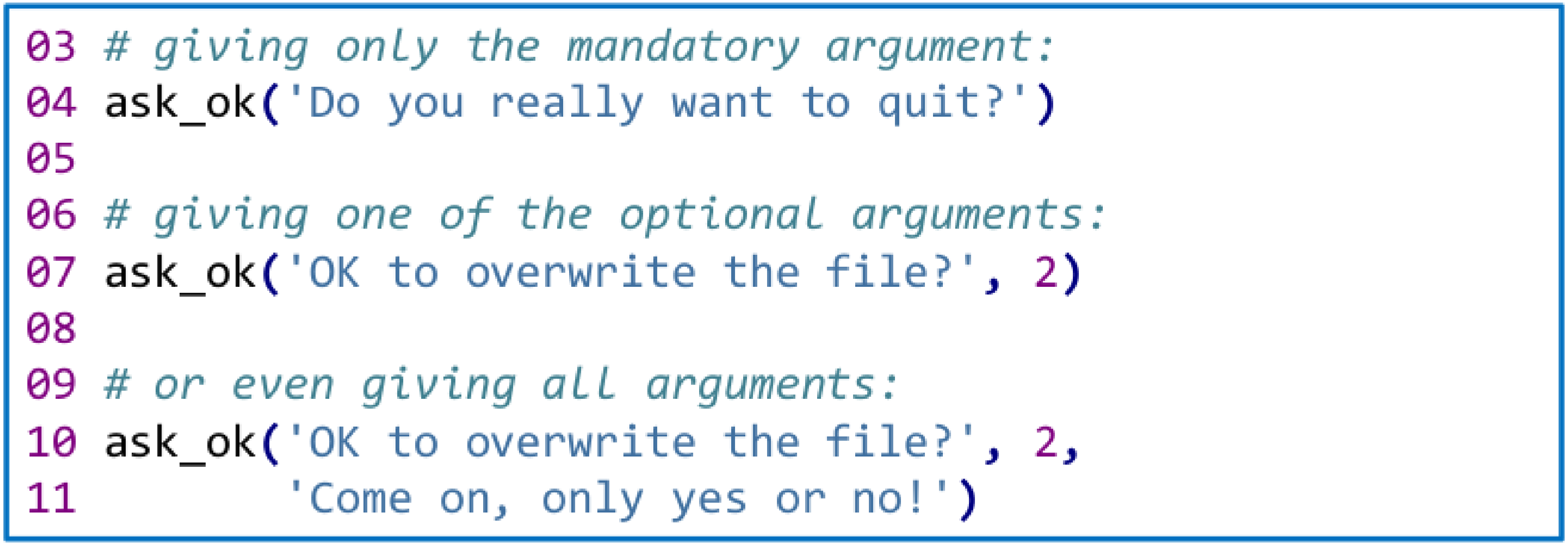


```python
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise OSError('uncooperative user')
        print(complaint)
```

สังเกตการใช้ **in** เพื่อตรวจสอบ ค่าที่สนใจ ว่าอยู่ในกลุ่มใด ๆ หรือไม่

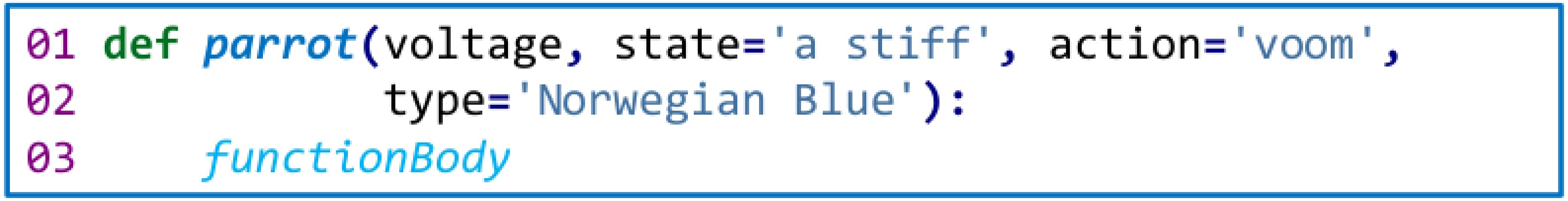
Keyword arguments and default value [4]

- So we can call this function with



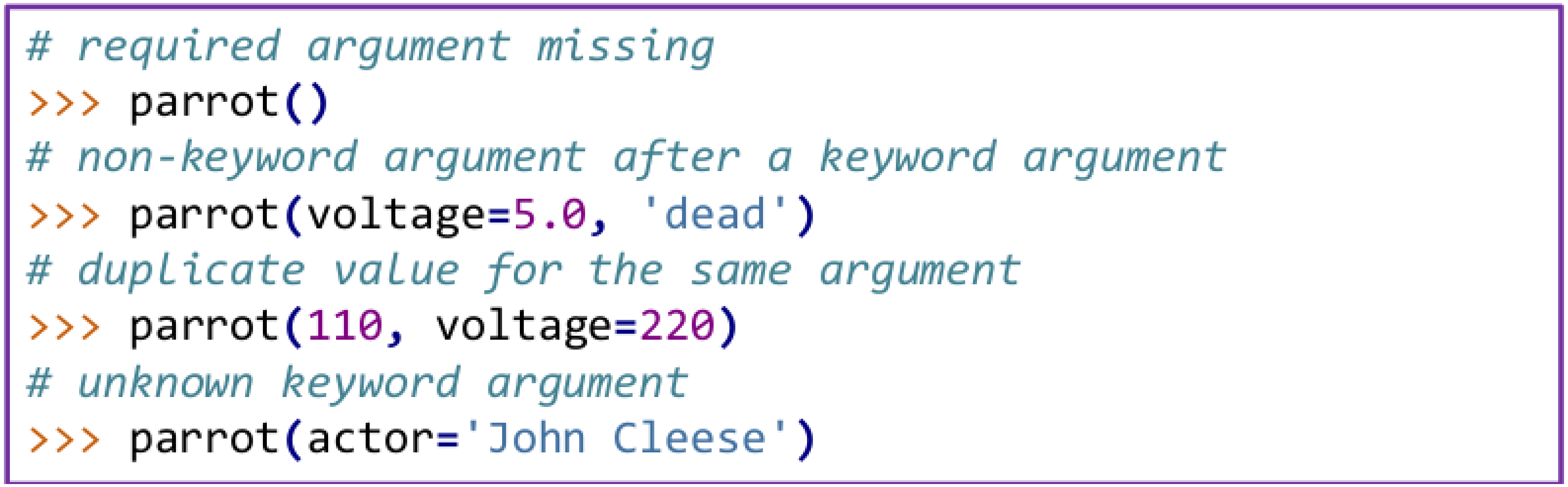
```
# giving only the mandatory argument:
ask_ok('Do you really want to quit?')

# giving one of the optional arguments:
ask_ok('OK to overwrite the file?', 2)

# or even giving all arguments:
ask_ok('OK to overwrite the file?', 2,
       'Come on, only yes or no!')
```

Keyword Arguments and Default Value [5]



```python
def parrot(voltage, state='a stiff', action='voom',
           type='Norwegian Blue'):
    functionBody
```

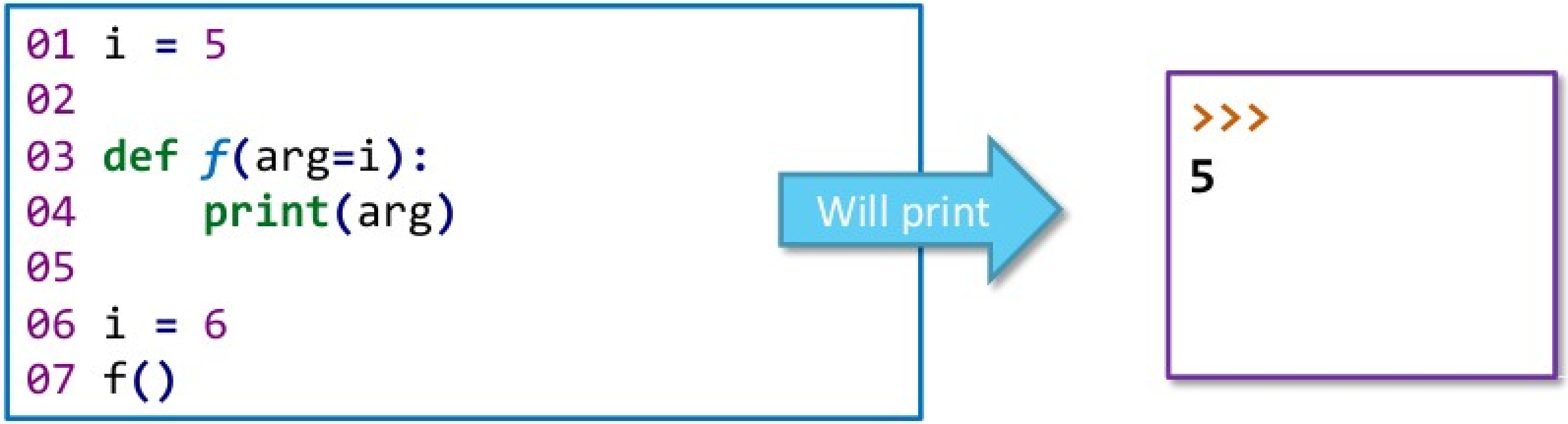
- The function accepts one required argument (voltage) and three optional arguments (state, action, and type).
- Invalid calls



```python
# required argument missing
>>> parrot()
# non-keyword argument after a keyword argument
>>> parrot(voltage=5.0, 'dead')
# duplicate value for the same argument
>>> parrot(110, voltage=220)
# unknown keyword argument
>>> parrot(actor='John Cleese')
```

Keyword Arguments and Default Value [6]

- The default values are evaluated at the point of function definition in the defining scope, so that

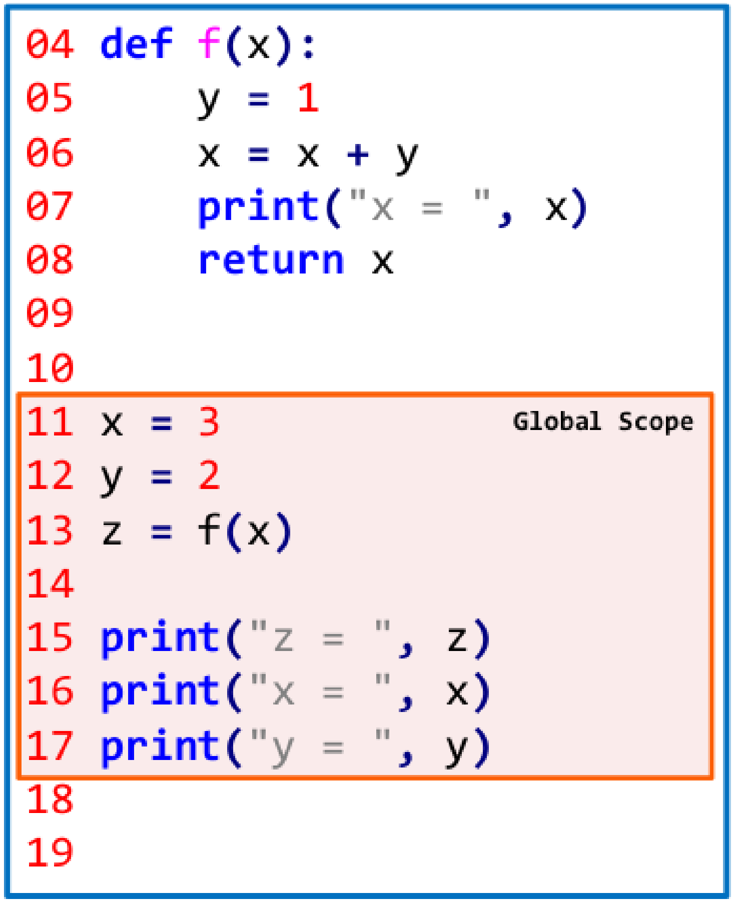


```python
i = 5

def f(arg=i):
    print(arg)

i = 6
f()
```

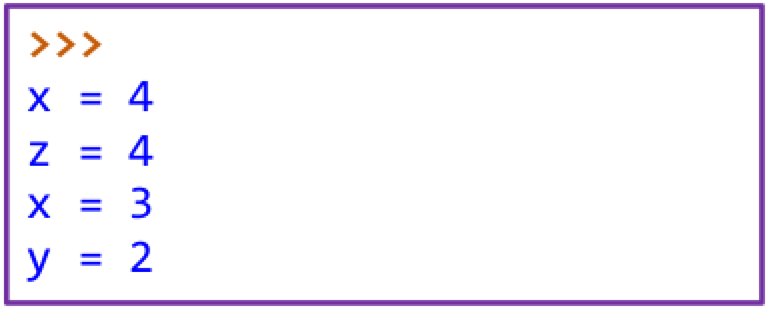
Will print

```
>>>
5
```

Variable scope



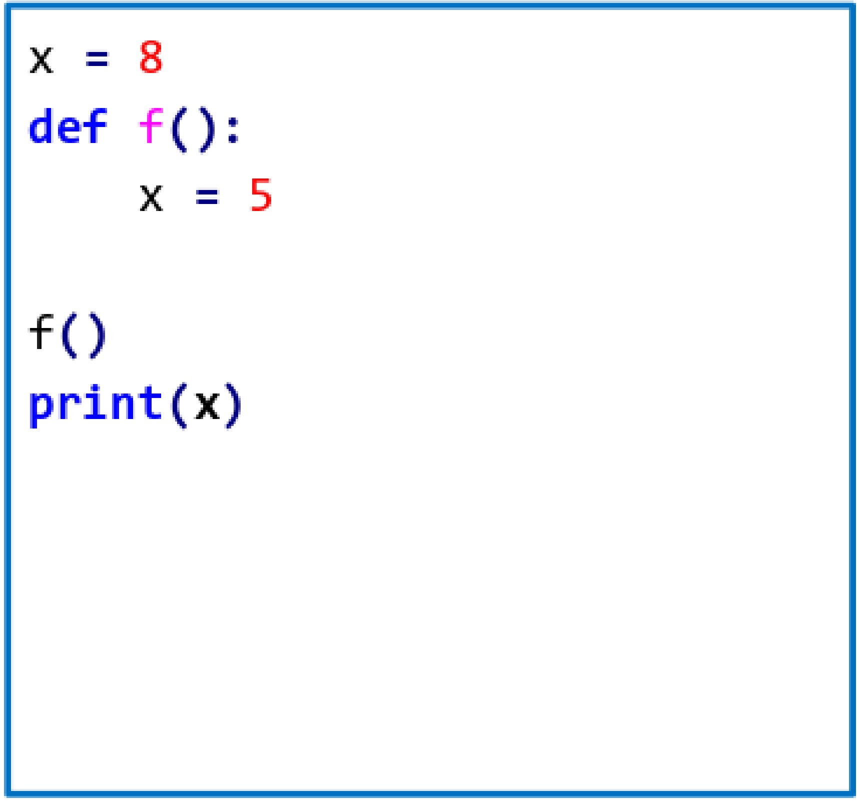
```
04 def f(x):
05     y = 1
06     x = x + y
07     print("x = ", x)
08     return x
09
10
11 x = 3
12 y = 2
13 z = f(x)
14
15 print("z = ", z)
16 print("x = ", x)
17 print("y = ", y)
18
19
```



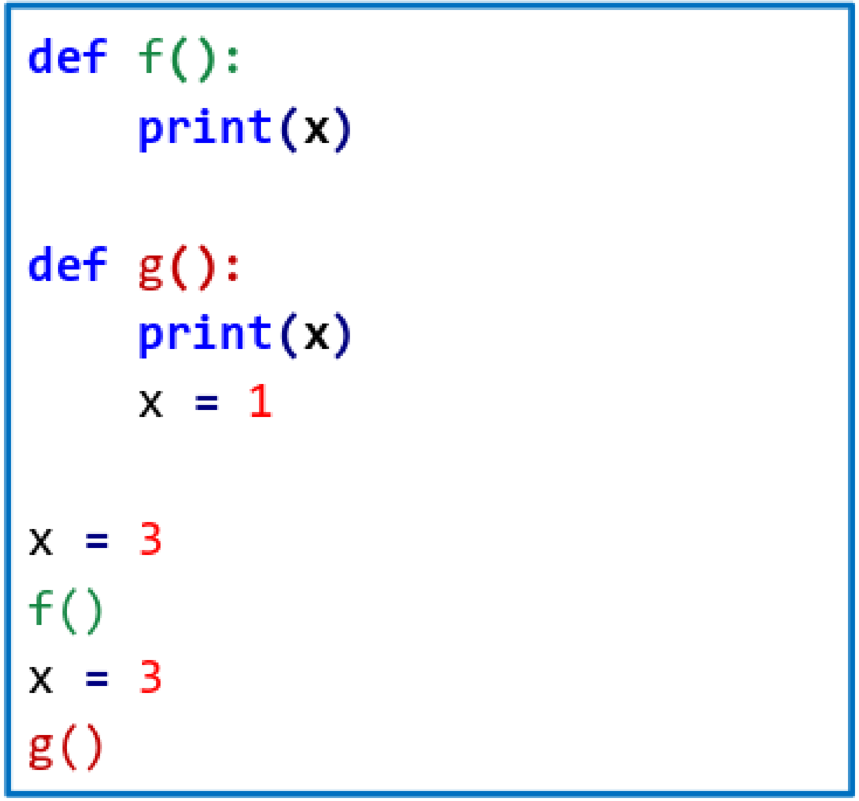
```
>>>
x = 4
z = 4
x = 3
y = 2
```

- Variable defined in f() is local variables
- modification of x and y in f() only has effect in f()
- y on line 5 and line 12 are different

Quick Quiz: Do these work ?



```python
x = 8
def f():
    x = 5

f()
print(x)
```

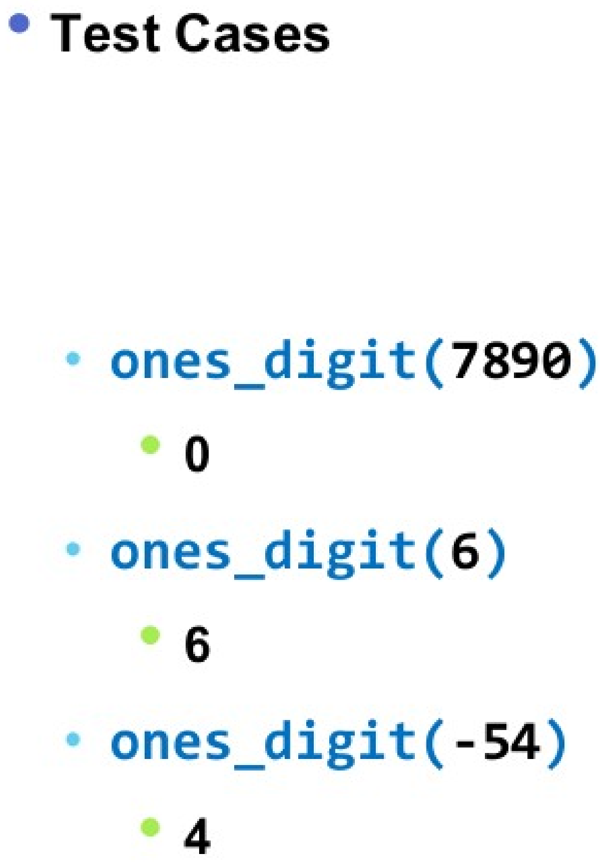


```python
def f():
    print(x)

def g():
    print(x)
    x = 1

x = 3
f()
x = 3
g()
```

Example 1: Ones Digit

- Question: Write a function that takes as input an integer and return right most digit of that number
- Analysis
  - How many (input) parameter ? Of which type ?
  - How many output ? Of which type ?
- We should begin with constructing testcases to make sure that we understand the problem

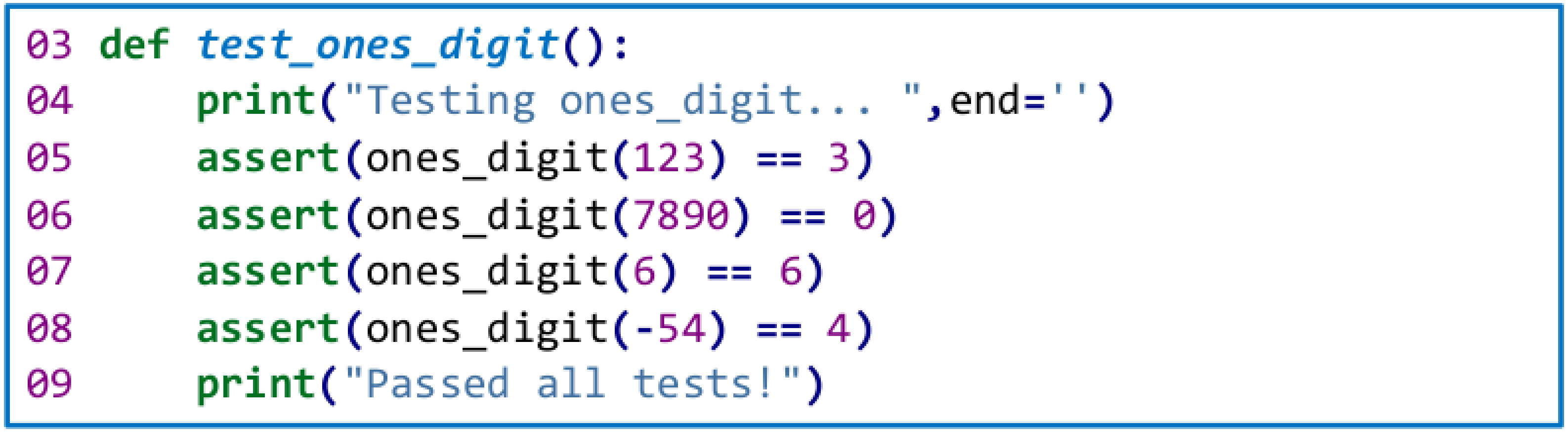
Example 1: Ones Digit [2]



- Test Cases
  - ones_digit(7890)
    - 0
  - ones_digit(6)
    - 6
  - ones_digit(-54)
    - 4

Example 1: Ones Digit [3]

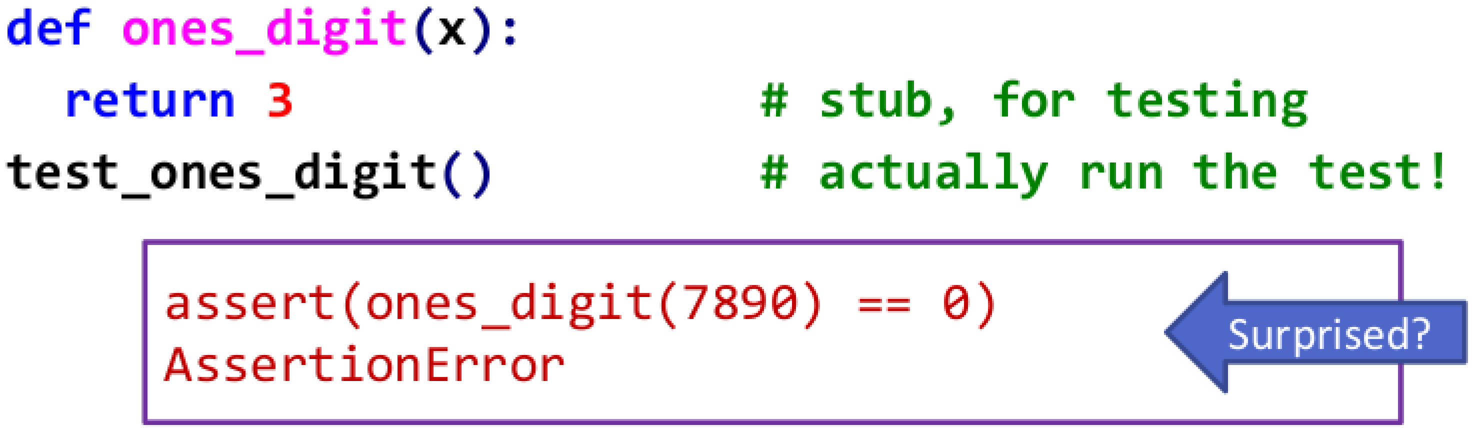
- We will write a function for *testing* ones_digit()
- We will use assert() built-in function for checking the result
- assert() will check if its argument is True, otherwise Exception will be thrown.



```
def test_ones_digit():
    print("Testing ones_digit... ",end='')
    assert(ones_digit(123) == 3)
    assert(ones_digit(7890) == 0)
    assert(ones_digit(6) == 6)
    assert(ones_digit(-54) == 4)
    print("Passed all tests!")
```

Example 1: Ones Digit [4]

- Start with stub solution so that we can run the test



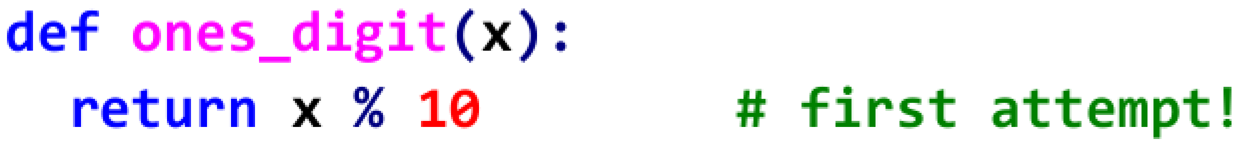
```
def ones_digit(x):
    return 3                    # stub, for testing
test_ones_digit()               # actually run the test!
```

```
assert(ones_digit(7890) == 0)
AssertionError
```

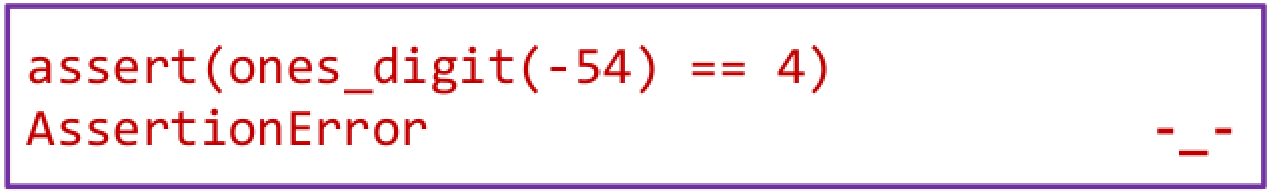
Surprised?

Example 1: Ones Digit [5]

- Now solve the problem, test and repeat
- How do we do that ?
  - the x % y gives the remainder
  - So the rightmost digit is just x % 10



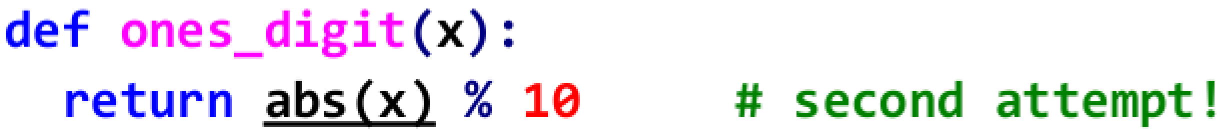
```python
def ones_digit(x):
  return x % 10        # first attempt!
```



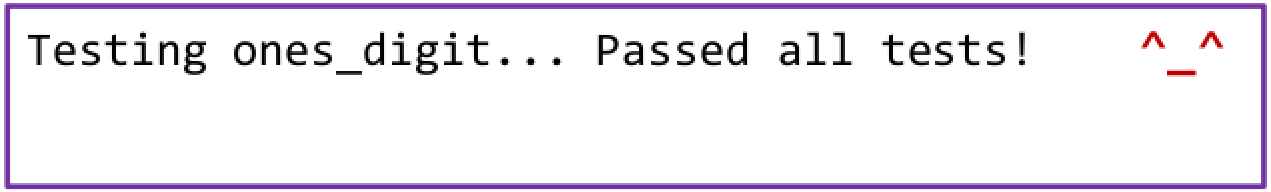
```
assert(ones_digit(-54) == 4)
AssertionError                            -_-
```

Example 1: Ones Digit [6]

- Solve, test, repeat
- Seems like our solution works on positive number but no the negative ones
- How about this ?



```python
def ones_digit(x):
  return abs(x) % 10      # second attempt!
```



```
Testing ones_digit... Passed all tests!    ^_^
```

Example 2: Sphere Volume

- Write a program to read the surface area of a sphere from user and calculate the volume of the sphere
- Analysis
  - Input ?
  - Output ?
- Try to use a lot of function