

# Introduction to Computer Science

- Dr. Ihsan Hussein

# Imperative Programming

- Python Programs
- Interactive Input/Output
- One-Way and Two-Way `if` Statements
- `for` Loops
- User-Defined Functions
- Assignments Revisited and Parameter Passing

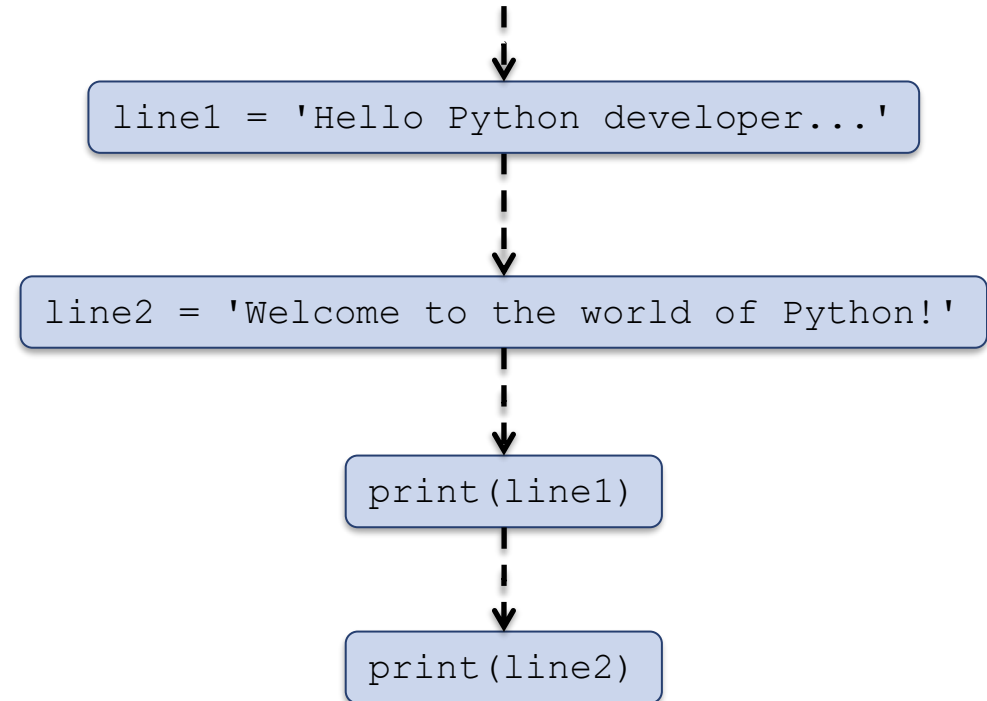
# Python program

A Python program is a sequence of Python statements

- Stored in a text file called a Python module
- Executed using an IDE or “from the command line”

```
line1 = 'Hello Python developer...'  
line2 = 'Welcome to the world of Python!'  
print(line1)  
print(line2)
```

hello.py



```
$ python hello.py  
Hello Python developer...
```

# Built-in function `print()`

Function `print()` prints its input argument to the IDLE window

- The argument can be any object: an integer, a float, a string, a list, ...
  - Strings are printed without quotes and “to be read by people”, rather than “to be interpreted by Python”,
- The “string representation” of the object is printed

```
>>> print(0)
0
>>> print(0.0)
0.0
>>> print('zero')
zero
>>> print([0, 1, 'two'])
[0, 1, 'two']
```

# Built-in function `input()`

Function `input()` requests and reads input from the user interactively

- It's (optional) input argument is the request message
- Typically used on the right side of an assignment statement

When executed:

1. The input request message is printed
2. The user enters the input
3. The *string* typed by the user is assigned to the variable on the left side of the assignment statement

```
>>> name = input('Enter your name: ')
Enter your name: Michael
>>>
```

```
first = input('Enter your first name: ')
last = input('Enter your last name: ')
line1 = 'Hello' + first + ' ' + last + '...'
print(line1)
print('Welcome to the world of Python!')
```

`input.py`

# Built-in function `eval()`

Function `input()` evaluates anything the user enters as a string

What if we want the user to interactively enter non-string input such as a number?

- Solution 1: Use type conversion
- Solution 2: Use function `eval()`
  - Takes a string as input and evaluates it as a Python expression

```
>>> age = input('Enter your age: ')
Enter your age: 18
>>> age
'18'
```

# Exercise

Write a program that:

1. Requests the user's name
2. Requests the user's age
3. Computes the user's age one year from now and prints the message shown

```
>>>  
Enter your name: Marie  
Enter your age: 17  
Marie, you will be 18 next year!
```

```
name = input('Enter your name: ')  
age = int(input('Enter your age: '))  
line = name + ', you will be ' + str(age+1) + ' next year!'  
print(line)
```

# Exercise

Write a program that:

1. Requests the user's name
2. Requests the user's age
3. Prints a message saying whether the user is eligible to vote or not

Need a way to execute a Python statement  
if a condition is true

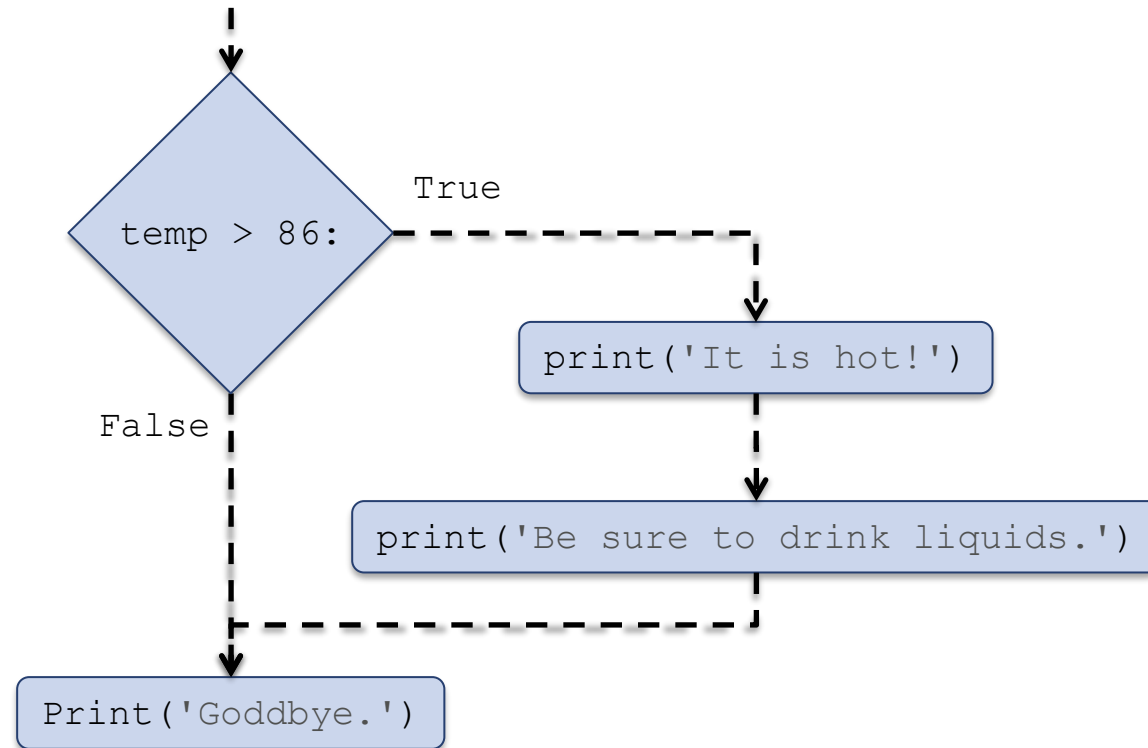


# One-way if statement

```
if <condition>:  
    <indented code block>  
<non-indented statement>
```

```
if temp > 86:  
    print('It is hot!')  
    print('Be sure to drink liquids.')  
print('Goodbye.')
```

The value of `temp` is 90.



# Exercises

Write corresponding if statements:

- a) If `age` is greater than 62 then print 'You can get Social Security benefits'
- b) If string 'large bonuses' appears in string `report` then print 'Vacation time!'
- c) If `hits` is greater than 10 and `shield` is 0 then print "You're dead..."

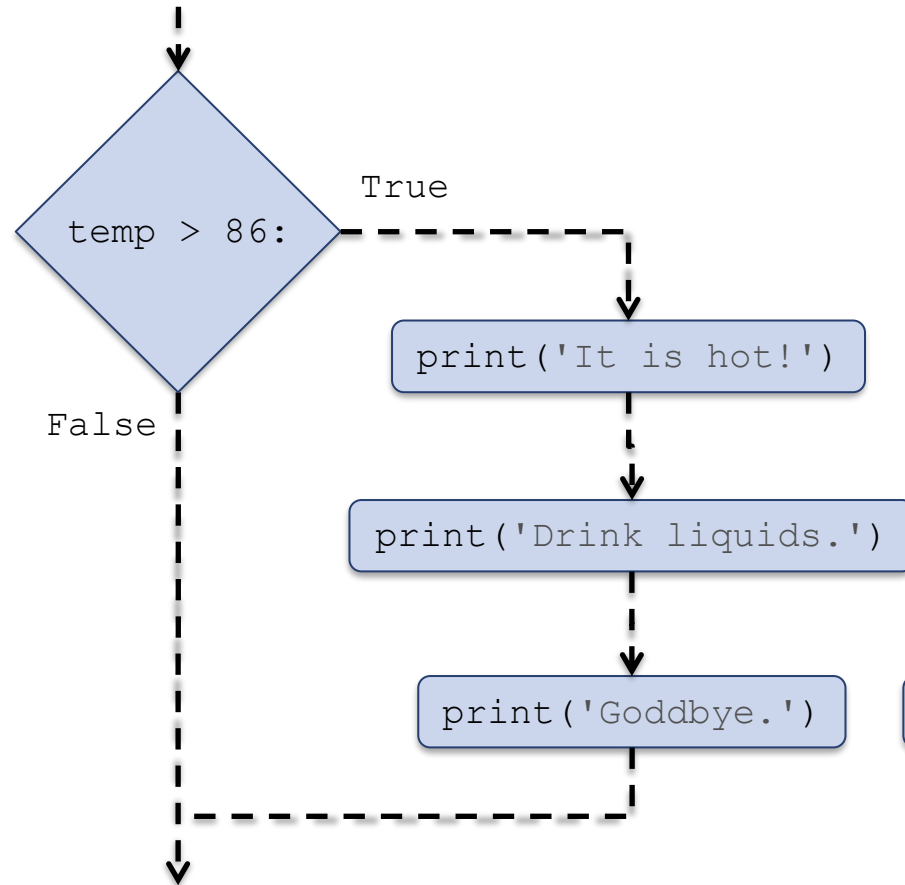
```
>>> hits = 12
>>> shield = 0
>>> if hits > 10 and shield == 0:
>>>     print("You're dead...")

You're dead...
>>> hits, shield = 12, 2
>>> if hits > 10 and shield == 0:
>>>     print("You're dead...")

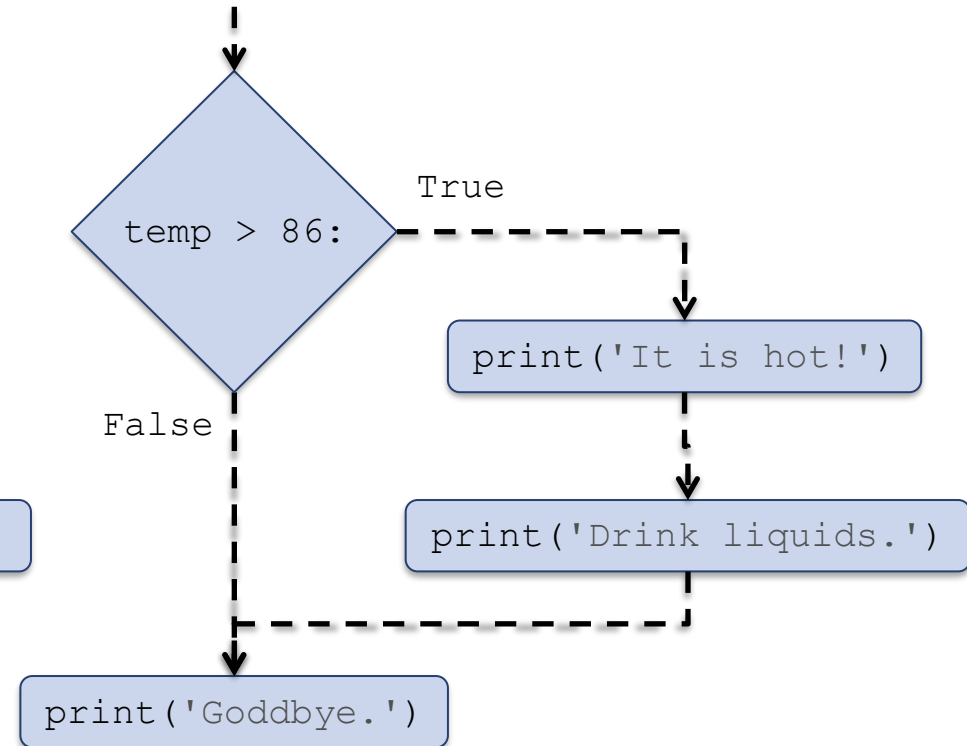
>>>
```

# Indentation is critical

```
if temp > 86:  
    print('It is hot!')  
    print('Drink liquids.')  
    print('Goodbye.')
```



```
if temp > 86:  
    print('It is hot!')  
    print('Drink liquids.')  
print('Goodbye.')
```

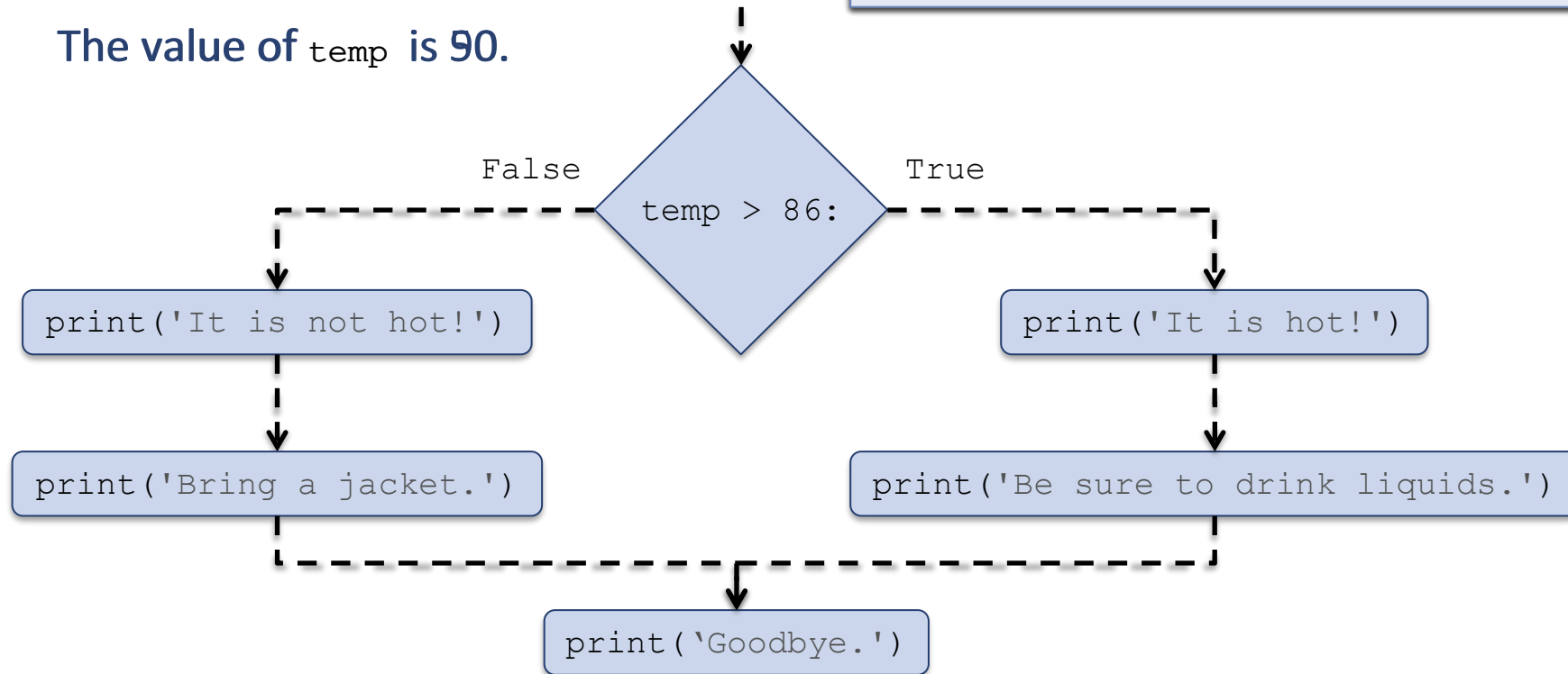


# Two-way if statement

```
if <condition>:  
    <indented code block 1>  
else:  
    <indented code block 2>  
<non-indented statement>
```

```
if temp > 86:  
    print('It is hot!')  
    print('Be sure to drink liquids.')  
else:  
    print('It is not hot.')  
    print('Bring a jacket.')  
print('Goodbye.')
```

The value of `temp` is 90.



# Exercise

Write a program that:

- 1) Requests the user's name
- 2) Requests the user's age
- 3) Prints a message saying whether the user is eligible to vote or not

```
name = input('Enter your name: ')
age = eval(input('Enter your age: '))
if age < 18:
    print(name + ", you can't vote.")
else:
    print(name + ", you can vote.")
```

```
>>>
Enter your name: Marie
Enter your age: 17
Marie, you can't vote.
>>>
=====RESTART=====
>>>
Enter your name: Marie
Enter your age: 18
Marie, you can vote.
>>>
```

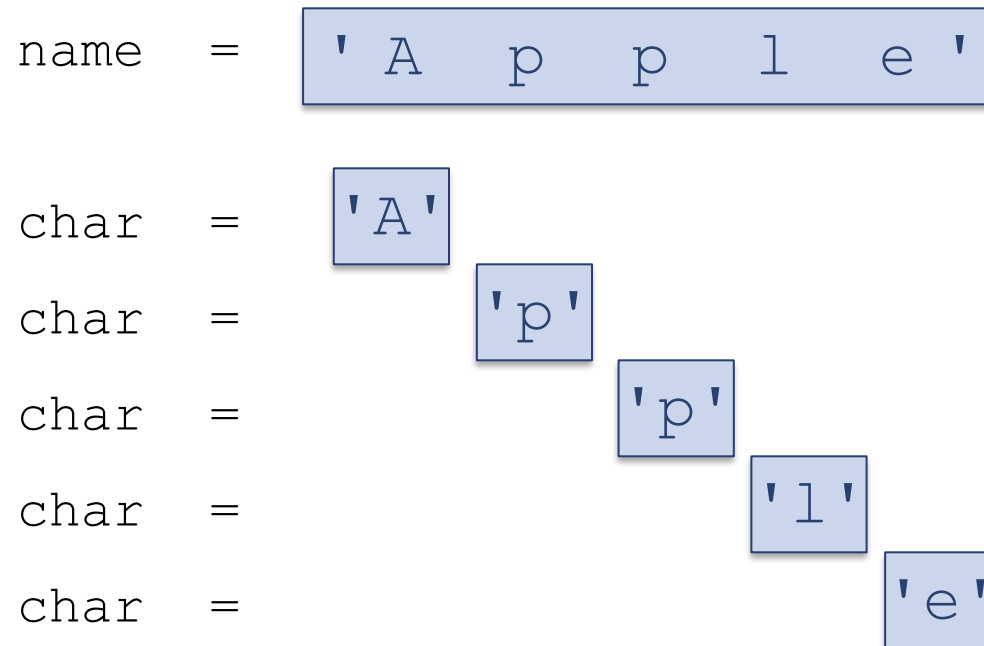
# Execution control structures

- The one-way and two-way if statements are examples of **execution control structures**
- **Execution control structures** are programming language statements that control which statements are executed, i.e., the execution flow of the program
- The one-way and two-way if statements are, more specifically, **conditional structures**
- **Iteration structures** are execution control structures that enable the repetitive execution of a statement or a block of statements
- The **for loop statement** is an iteration structure that executes a block of code for every item of a sequence

# for loop

Executes a block of code for every item of a sequence

- If sequence is a string, items are its characters (single-character strings)



```
>>> name = 'Apple'
>>> for char in name:
    print(char)
```

```
A
p
p
l
```

# for loop

Executes a code block for every item of a sequence

- Sequence can be a string, a list, ...
- Block of code must be indented

```
for <variable> in <sequence>:  
    <indented code block >  
<non-indented code block>
```

```
for word in ['stop', 'desktop', 'post', 'top']:  
    if 'top' in word:  
        print(word)  
print('Done.')
```

word = 'stop'

word = 'desktop'

word = 'post'

word = 'top'

```
>>>  
stop  
desktop  
top  
Done.
```



# Exercise

Write a “spelling” program that:

- 1) Requests a word from the user
- 2) Prints the characters in the word from left to right, one per line

```
name = input('Enter a word: ')
print('The word spelled out: ')

for char in name:
    print(char)
```

```
=====RESTART=====
>>>
Enter a word: omnipotent
The word spelled out:
o
m
n
i
p
o
t
e
n
t
>>>
```

# Built-in function `range()`

Function `range()` is used to iterate over a sequence of numbers in a specified range

- To iterate over the  $n$  numbers  $0, 1, 2, \dots, n-1$   
`for i in range(n):`
- To iterate over the  $n$  numbers  $i, i+1, i+2, \dots, n-1$   
`for i in range(i, n):`
- To iterate over the  $n$  numbers  $i, i+c, i+2c, i+3c, \dots, n-1$   
`for i in range(i, n):`

```
>>> for i in range(2, 16, 10):  
    print(i)
```

```
2  
12  
>>>
```

# Exercise

Write for loops that will print the following sequences:

- a) 0, 1, 2, 3, 4, 5, 6, 7, 8 , 9, 10
- b) 1, 2, 3, 4, 5, 6, 7, 8, 9
- c) 0, 2, 4, 6, 8
- d) 1, 3, 5, 7, 9
- e) 20, 30, 40, 50, 60

# Defining new functions

A few built-in functions we have seen:

- `abs()`, `max()`, `len()`,  
`sum()`, `print()`

New functions can be defined using `def`

`def`: function definition keyword

`f`: name of function

`x`: variable name for input argument

```
def f(x):  
    res = x**2 + 10  
    return res
```

`return`: specifies function output

```
>>> abs(-9)  
9  
>>> max(2, 4)  
4  
>>> lst = [2, 3, 4, 5]  
>>> len(lst)  
4  
>>> sum(lst)  
14  
>>> print()  
  
>>> def f(x):  
        res = 2*x + 10  
        return x**2 + 10  
  
>>> f(1)  
11  
>>> f(3)  
19  
>>> f(0)  
10
```

# print() versus return

```
def f(x):  
    res = x**2 + 10  
    return res
```

```
>>> f(2)  
14  
>>> 2*f(2)  
28
```

Function returns value of `res` which can then be used in an expression

```
def f(x):  
    res = x**2 + 10  
    print(res)
```

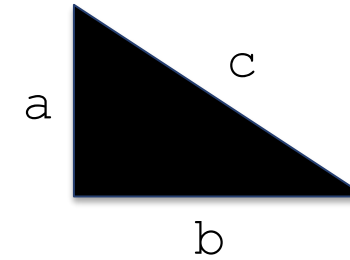
```
>>> f(2)  
14  
>>> 2*f(2)  
14  
Traceback (most recent call last):  
  File "<pyshell#56>", line 1, in  
<module>  
    2*f(2)  
TypeError: unsupported operand  
type(s) for *: 'int' and  
'NoneType'
```

Function prints value of `res` but does not return anything

# Defining new functions

The general format of a function definition is

```
def <function name> (<0 or more variables>):  
    <indented function body>
```



Let's develop function `hyp()` that:

- Takes two numbers as input (side lengths `a` and `b` of above right triangle )
- Returns the length of the hypotenuse `c`

```
>>> hyp(3,4)  
5.0  
>>>
```

```
import math  
def hyp(a, b):  
    res = math.sqrt(a**2 + b**2)  
    return res
```

# Exercise

Write function `hello()` that:

- takes a name (i.e., a string) as input
- prints a personalized welcome message

Note that the function does not return anything

```
>>> hello('Julie')
Welcome, Julie, to the world of Python.
>>>
```

```
def hello(name):
    line = 'Welcome, ' + name + ', to the world of Python.'
    print(line)
```

# Exercise

Write function `rng()` that:

- takes a list of numbers as input
- returns the range of the numbers in the list

The range is the difference between the largest and smallest number in the list

```
>>> rng([4, 0, 1, -2])  
6  
>>>
```

```
def rng(lst):  
    res = max(lst) - min(lst)  
    return res
```



# Comments and docstrings

Python programs should be documented

- So the developer who writes/maintains the code understands it
- So the user knows what the program does

## Comments

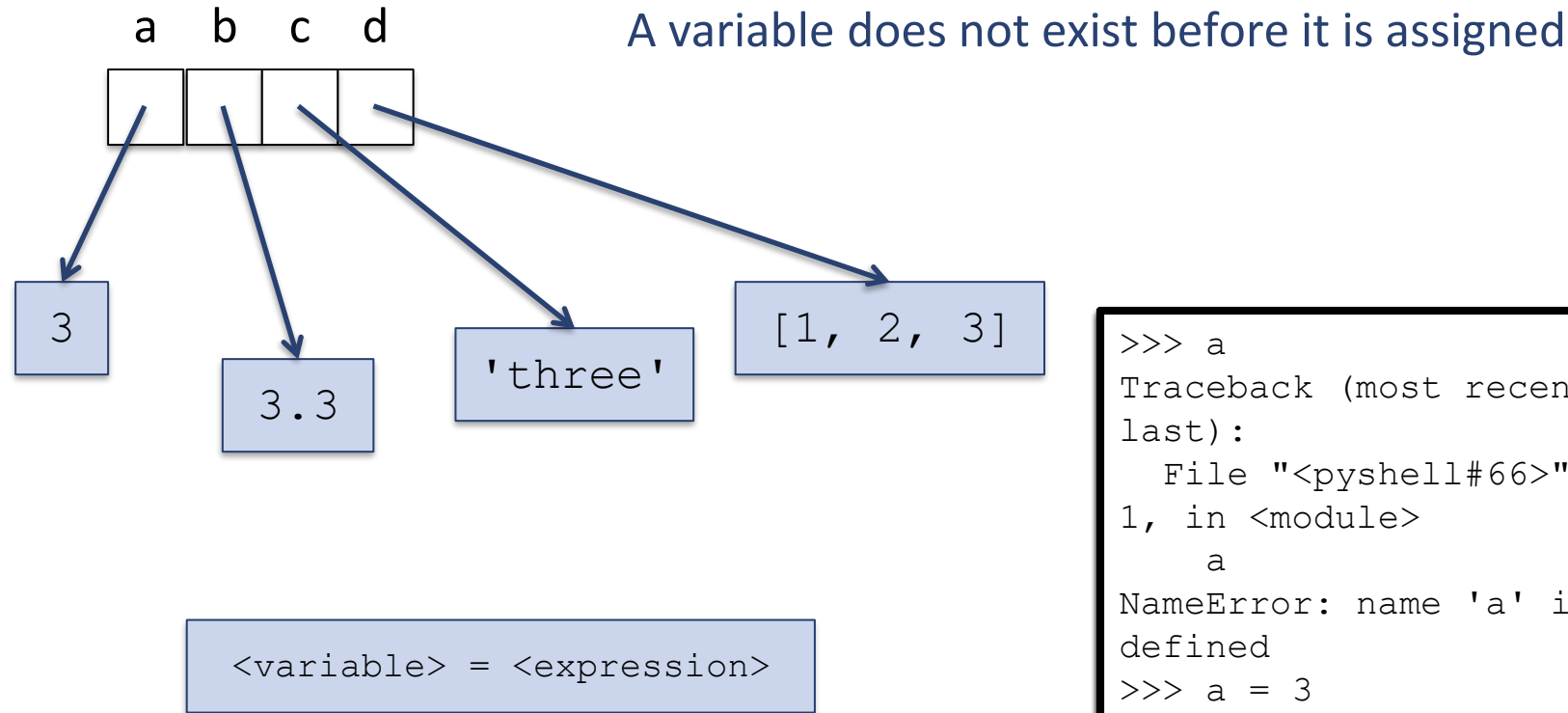
```
def f(x):  
    res = x**2 + 10    # compute result  
    return res         # and return it
```

## Docstring

```
def f(x):  
    'returns x**2 + 10'  
    res = x**2 + 10    # compute result  
    return res         # and return it
```

```
>>> help(f)  
Help on function f in module  
__main__:  
  
f(x)  
  
>>> def f(x):  
        'returns x**2 + 10'  
        res = x**2 + 10  
        return res  
  
>>> help(f)  
Help on function f in module  
__main__:  
  
f(x)  
    returns x**2 + 10  
  
>>>
```

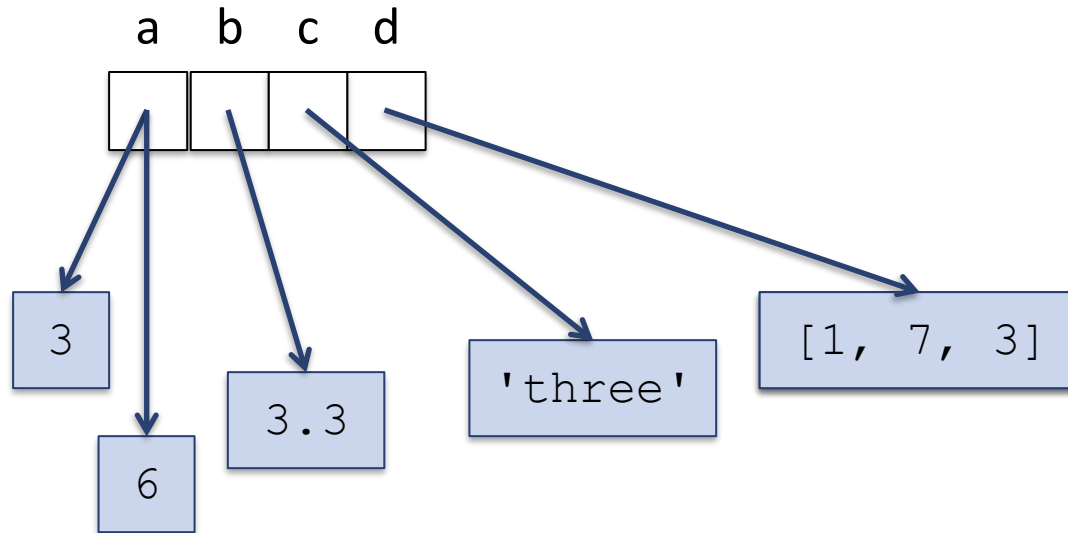
# Assignment statement: a second look



1. `<expression>` is evaluated and its value put into an object of appropriate type
2. The object is assigned name `<variable>`

```
>>> a
Traceback (most recent call
last):
  File "<pyshell#66>", line
1, in <module>
    a
NameError: name 'a' is not
defined
>>> a = 3
>>> b = 2 + 1.3
>>> c = 'three'
>>> d = [1, 2] + [3]
```

# Mutable and immutable types



The object (3) referred to by variable `a` does not change; instead, `a` refers to a new object (6)

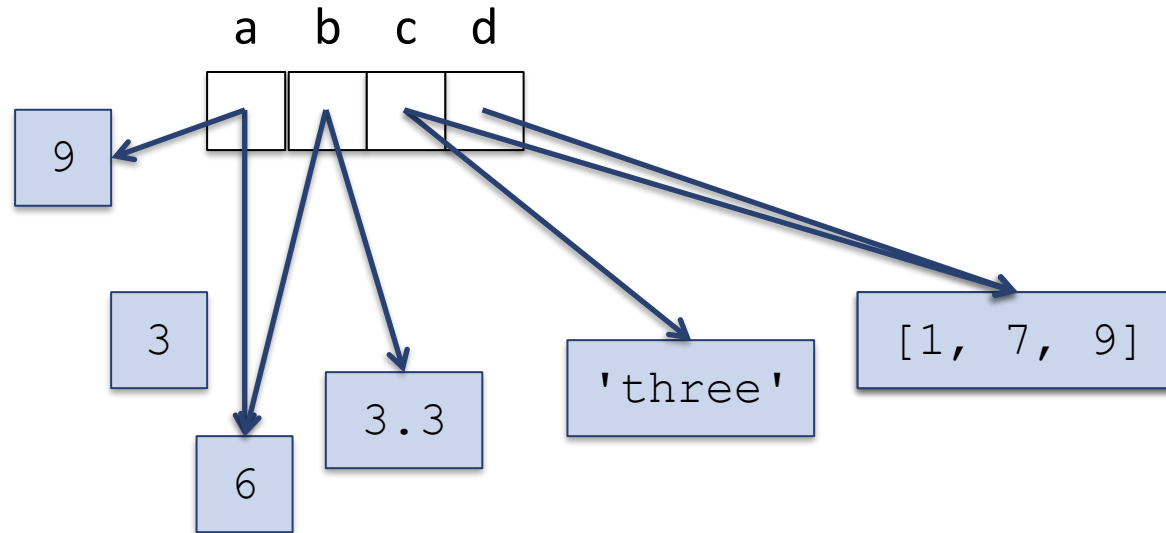
- Integers are **immutable**

The object (`[1, 2, 3]`) referred to by `d` changes

- Lists are **mutable**

```
>>> a
3
>>> a = 6
>>> a
6
>>> d
[1, 2, 3]
>>> d[1] = 7
>>> d
[1, 7, 3]
```

# Assignment and mutability



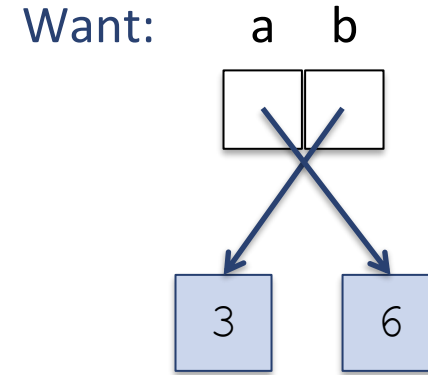
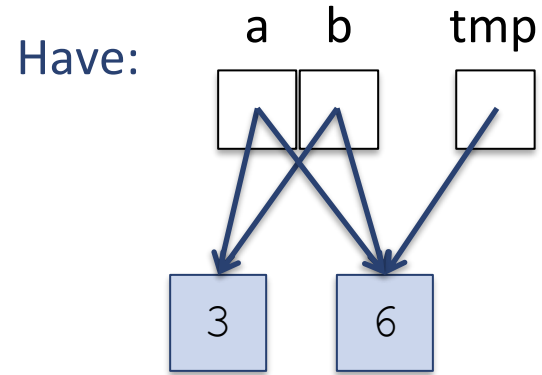
**a and b refer to the same integer object**

```
>>> a
6
>>> b
3.3
>>> b = a
>>> b
6
>>> a = 9
>>> b
6
>>> c = d
>>> c
[1, 7, 3]
>>> d[2] = 9
>>> c
[1, 7, 9]
```

**The list that c refers to changes(9); b still refers to the old object(6) so it changes too**

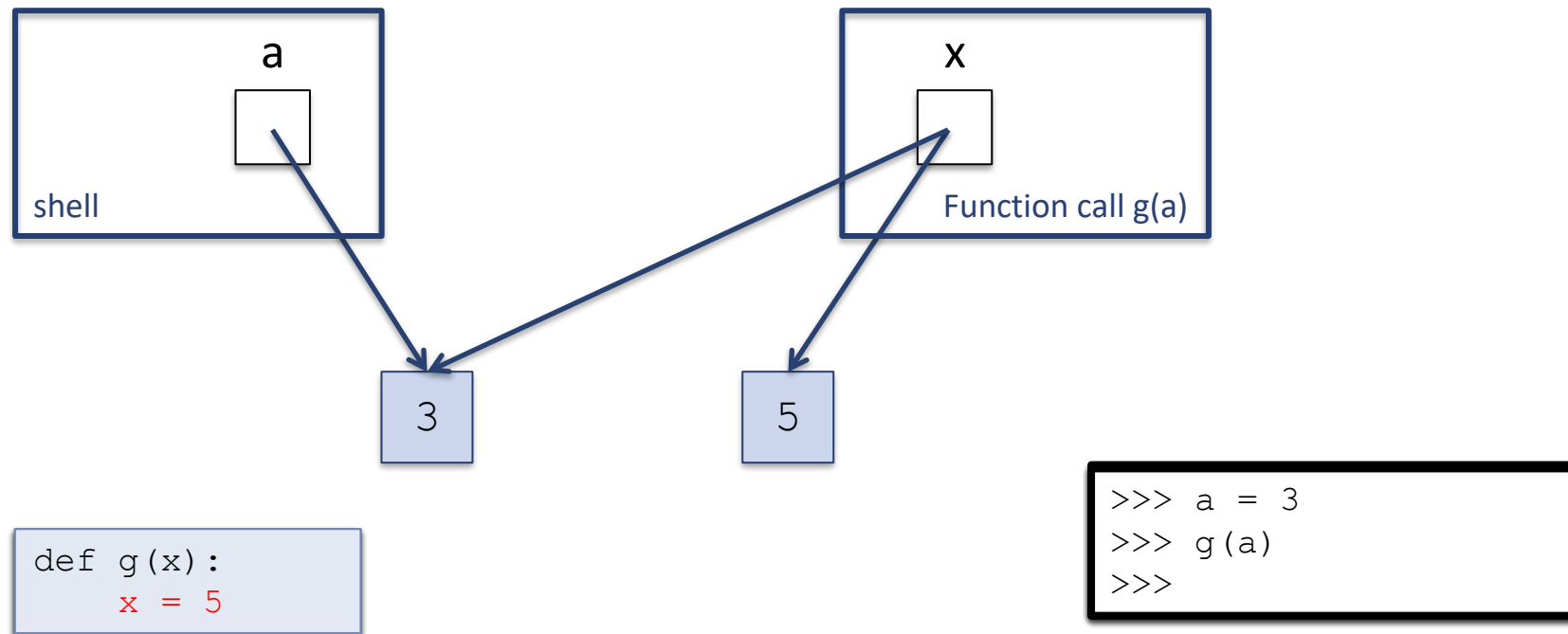
- Because integers are immutable, changing a does not affect the value of b

# Swapping values



```
>>> a
3
>>> b
6
>>> tmp = b
>>> b = a
>>> a = tmp
```

# Immutable parameter passing

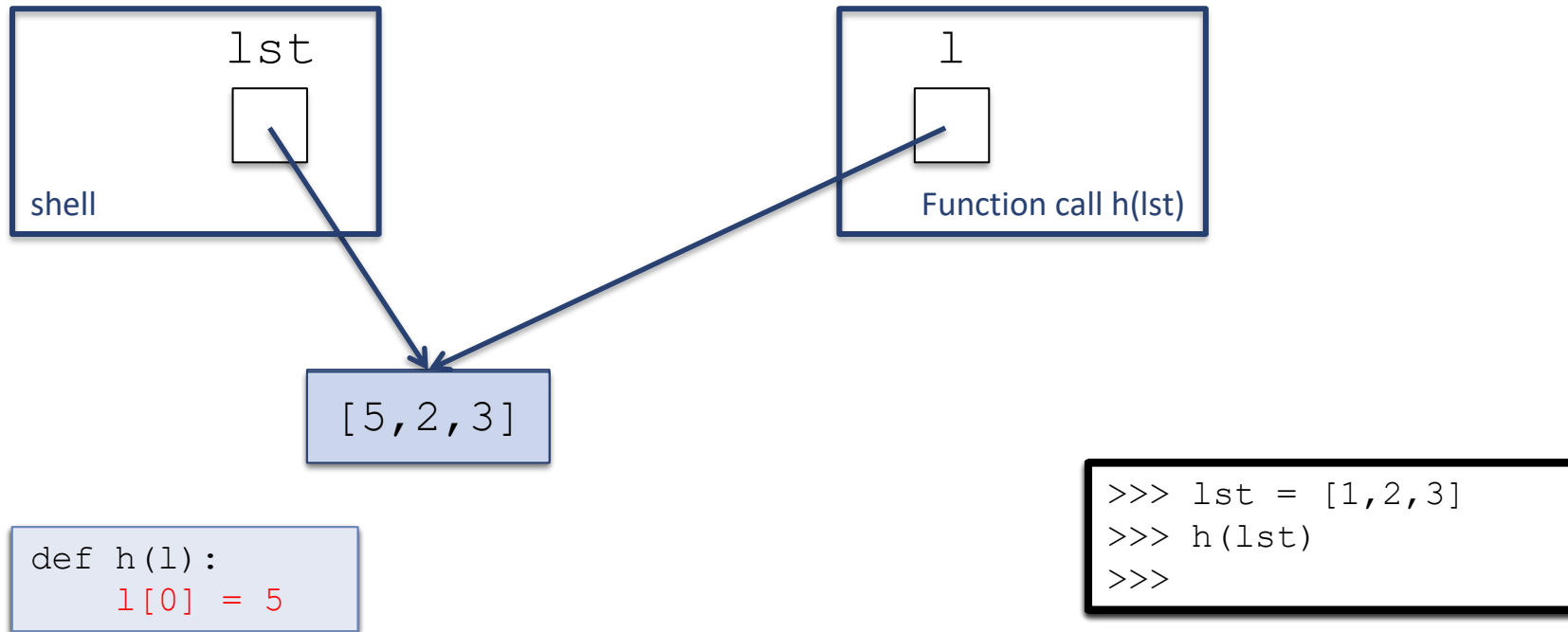


Variable `x` inside `g()`, refers to the object `3` as the value of `a` in the interactive shell.

As if we executed `x = a`

This is because `a` refers to an immutable object.

# Mutable parameter passing



Variable `l` inside `h()` refers to the object `lst` in the interactive shell.

This is because `lst` and `l` refer to an mutable object.

# Exercise

Write function `swapFS()` that:

- takes a list as input
- swaps the first and second element of the list, but only if the list has at least two elements

The function does not return anything

```
>>> mylst = ['one', 'two', 'three']
>>> swapFS(mylst)
>>> mylst
['two', 'one', 'three']
>>> mylst = ['one']
>>> swapFS(mylst)
>>> mylst
['one']
>>>
```

```
def swapFS(lst):
    if len(lst) > 1:
        lst[0], lst[1] = lst[1], lst[0]
```



Thank You