METU Computer Engineering

Ceng 111 – Fall 2018
Week 8

Actions, Functions

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Today

- Side effect
- Input/Output Actions
- Basic Statements
- Compound Statements
- Functions

Administrative Notes

- Live sessions schedule change
  - Tue 13:40 Session (a.k.a. common session)
  - Wed 10:40 Session
  - ~~Wed 15:40: Section 2~~
  - ~~Thu 15:40: Section 1~~
- Social session
- The labs
- THE1
- Midterm: 11 December 17:40

Side effect

```
1 def f(L):
2       L[0] += 2
3       return L[0]
4
5 M=[2, 3, 4]
6 x = f(M) + M[0]
```

Evaluation order yields two different results

Evaluate f(M) first

Evaluate M[0] first

Output in Python

```
>>> print "I am %f tall, %d years old and have %s eyes" % (1.86, 20, "brown")
I am 1.860000 tall, 20 years old and have brown eyes
```

- %f → Data identifier
- We have the following identifiers in Python:

Identifier	Description
d, i	Integer
f, F	Floating point
e, E	Floating point in exponent form
s	Using the `str()` function
r	Using the `repr()` function
%	The % character itself

Output in Python

```
>>> print "I am {0} tall, {1} years old and have {2} eyes".format(1.86, 20, "brown")
I am 1.86 tall, 20 years old and have brown eyes
```

- {0}, {1}, {2} → Data fields
- Instead of numbers, we can give names to the fields:

```
>>> print "I am {height} tall, {age} years old and have {color} eyes".\
        format(height=1.86, age=20, color="brown")
I am 1.86 tall, 20 years old and have brown eyes
```

- We can re-use the fields

```
>>> print "I am {height} tall, {age} years old. I am {height} tall.".\
        format(age=20,height=1.86)
I am 1.86 tall, 20 years old. I am 1.86 tall.
```

METU Computer Engineering

Basic Statements

Examples:

del L[2]

print “this is a string”

Compound Statements

- Involves more than one expression or statement
- Example:

$$\textbf{if } \beta \textbf{ then } \sigma$$

$$\textbf{if } \beta \textbf{ then } \sigma_1 \textbf{ else } \sigma_2$$

$$\textbf{while } \beta \textbf{ do } \sigma$$

$$\textbf{for } v = 1 \textbf{ to } 5 \textbf{ do print } v, v * (v - 1)$$

Conditional Statements

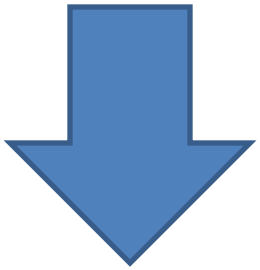
if $\langle boolean\ \ expression \rangle$ **then** $\langle action \rangle$

Translated to:

compute the $\langle boolean\ \ expression \rangle$, leave the result in the relevant register r
branch to α if $r \stackrel{?}{=} 0$
carry out $\langle action \rangle$
α : $\langle some\ actions\ that\ follow\ the\ \mathbf{if} \rangle$

Conditional Statements

if $\langle boolean\ \ expression \rangle$ **then** $\langle action_{TRUE} \rangle$
if $\neg \langle boolean\ \ expression \rangle$ **then** $\langle action_{FALSE} \rangle$



if $\langle boolean\ \ expression \rangle$ **then** $\langle action_{TRUE} \rangle$
else $\langle action_{FALSE} \rangle$

Conditional Statements in Python

```
1 if <condition-expression> :
2         <statements-1>
3 else:
4         <statements-2>
```

- the syntax is important!
- indentation is extremely important!
- “else”-part can be omitted!

You can indent your Python code using tabs or space. However, it is a good programming practice to use only one of them while indenting your code: *i.e.*, do not mix them!

Multiple If Statements in Python

```
1  if <condition-expression-1> :
2          <statements>
3  elif <expression-2> :
4          <statements>
5  .
6  .
7  .
8  elif <expression-M> :
9          <statements>
10 else:
11         <statements>
```

Multiple **Nested** If Statements in Python

```
1 if <condition-expression-1> :
2         <statements-1>
3         if <condition-expression-2>:
4                 <statements-2>
5         else:
6                 <statements-3>
7 else:
8         <statements-4>
```

Conditional Expression in Python

<exp-1> if <cond-exp> else <exp-2>

Note that this is an expression not a statement!!

METU Computer Engineering

Functions: Reusable Actions

- In programming, we often combine the statements that we use frequently together into functions.

```
1   void main()
2   {
3       hello();
4
5       ... // Some execution here
6
7       hello();
8   }
9
10  void hello()
11  {
12      ...
13      // I am looong function involving lots and lots of statements
14      ...
15  }
```

Functions: Reusable Actions (cont'd)

- Functions in programming are similar to functions in Mathematics but there are differences.
- Difference to mathematical functions:
  - A function in programming may not return a value.
  - A function in mathematics only depends on its arguments unlike the functions in programming.
  - A mathematical function does not have the problem of side effects.

Functions: Reusable Actions

- Why do we need functions?
  - Reusability
  - Structure
  - Other benefits of the functional paradigm

Functions in Python

```
1 def function-name(parameter-1, ..., parameter-N):
2         statement-1
3         .
4         .
5         statement-M
```

- Syntax is important!
- Indentation is extremely important

Functions in Python

- Write a Python function that reverses a given number
  - Example: If 123 is given, the output should be 321

```
1  def reverse(a):
2      return int(str(a)[::-1])
3
```

Nested Functions in Python

```
1 def f(N):
2         Number = N
3         def g():
4                 C = 20
5                 return N * Number
6         print "Number", N, "and its square:", g()
```

- Function `g()` can access all the local variables as well as the parameters of function `f()`.
- Function `f()` cannot access the local variables of function `g()`!
- Function `g()` cannot be used before it is defined! For example, the second line could not have been `Number = 10 * g(10)`.
- The indentation is extremely important to understand which statement belongs to which function! For example, the last line is part of function `f()` since they are at the same indentation!

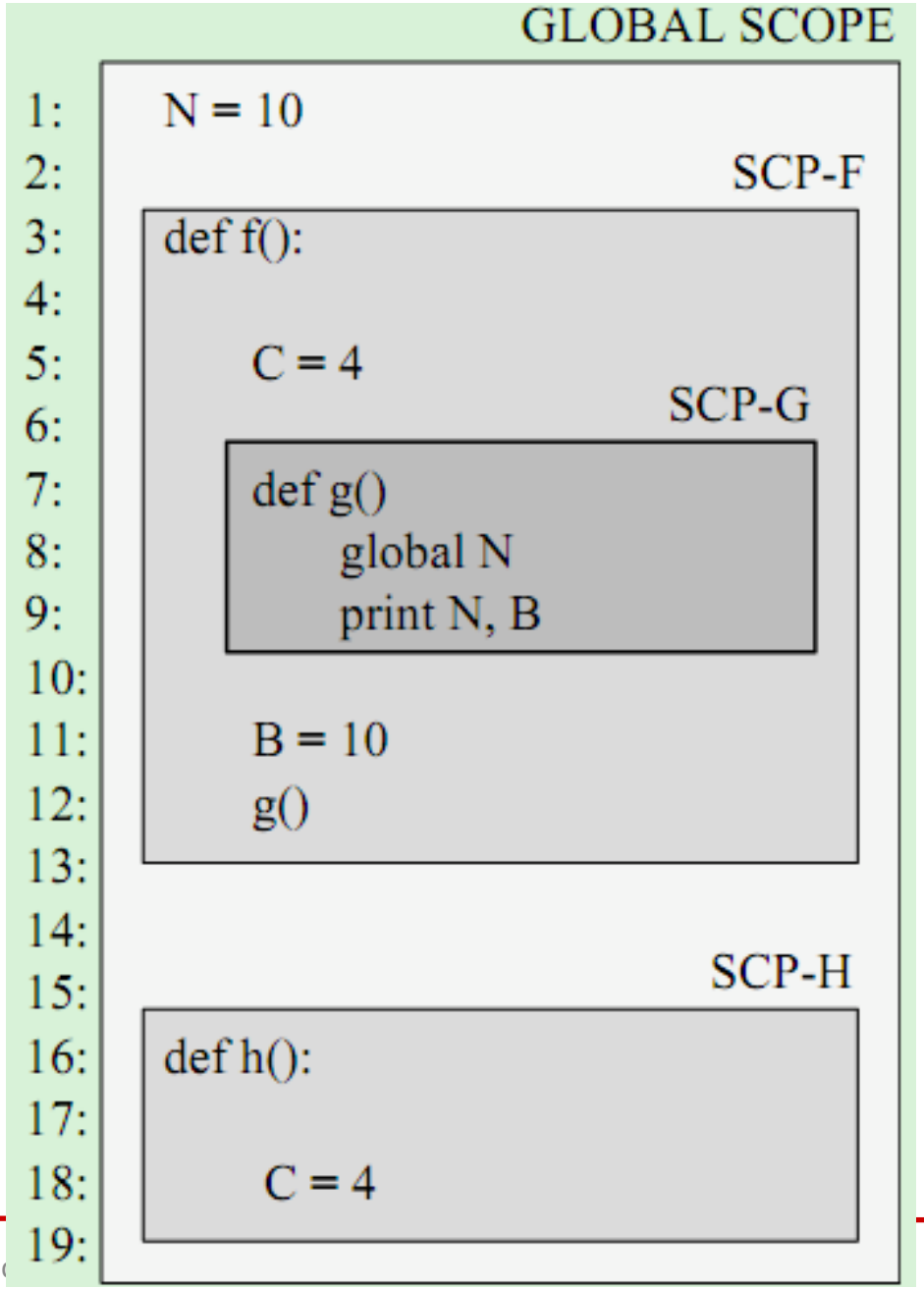
Global Variables in Python

- To access variables in the global workspace, you should use “global <varname>”

```
1 N = 10
2 def f():
3         global N
4         def g(Number):
5                 C = 20
6                 return N * Number
7         N = g(N)
```

Scope in Python

- Since you can nest functions in Python, understanding scope is important



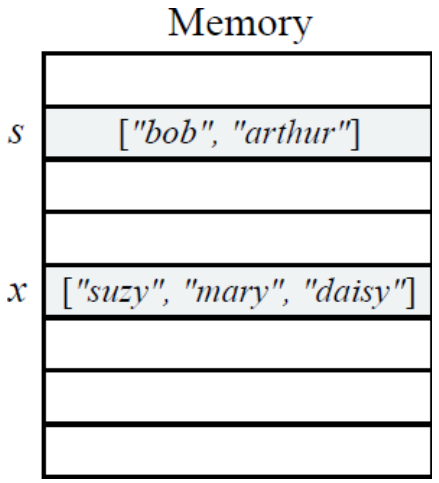
METU Computer Engineering

Parameter passing in functions

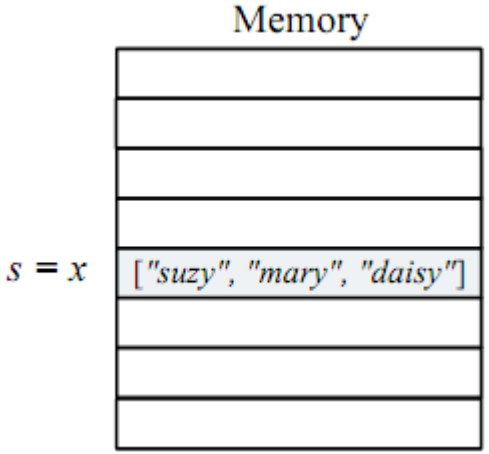
```
define f(x)
    x[0] ← "jennie"
    x ← ["suzy","mary","daisy"]
    return x

...
s ← ["bob","arthur"]
print f(s)
print s
...
```

(a) `["suzy","mary","daisy"]`
`["bob","arthur"]`

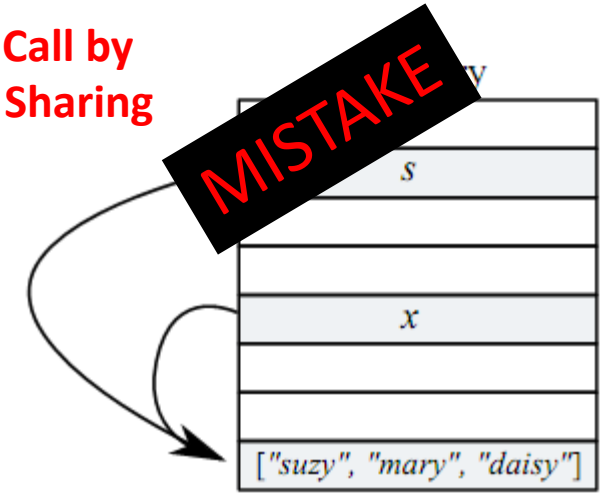
Call by Value



Call by Reference



Call by Sharing



(b) `["suzy","mary","daisy"]`
`["suzy","mary","daisy"]`

(c) `["suzy","mary","daisy"]`
`["jennie","arthur"]`

Parameter passing in functions in Python

```
1 def f(N):
2         N = N + 20
3
4 def g():
5         A = 10
6         print A
7         f(A)
8         print A
```

```
>>> g()
10
10
```

METU Computer Engineering

Parameter passing in functions in Python

```
1 def f(List):
2         List[0] = 'A'
3
4 def g():
5         L = [1, 2, 3]
6         print L
7         f(L)
8         print L
```

```
>>> g()
[1, 2, 3]
['A', 2, 3]
```

METU Computer Engineering

Parameter passing in functions in Python

```
1 def f(List):
2         List = List[::-1]
3
4 def g():
5         L = [1, 2, 3]
6         print L
7         f(L)
8         print L
```

```
>>> g()
[1, 2, 3]
[1, 2, 3]
```

METU Computer Engineering

Default Parameters in Python

```
1 def reverse_num(Number=123):
2   """reverse_num: Reverse
3                   the digits in a number"""
4   str_num = str(Number)
5   print "Reverse of", Number, "is", str_num[::-1]
```

- We can now call this function with reverse_num() in which case Number is assumed to be 123.
- If we supply a value for Number, that value is used instead.

```
1 def f(Str, Number=123, Bst="Some"):
2   print Str, Number, Bst
```

While we are at it…

let's have a look at commenting in Python

```
1 def reverse_num(Number=123):
2   """reverse_num: Reverse
3                   the digits in a number"""
4   str_num = str(Number)
5   print "Reverse of", Number, "is", str_num[::-1]
```

There are two different ways to put comments in Python: (1) You can use # in which case the rest of the line is not interpreted. (2) You can enclose multiple lines like """ <lines of text> """. The comments that are written using the second option are basically documentation strings and available through the `help` page.

Functional Programming in Python

METU Computer Engineering

- List Comprehension

```
[<expr> for <var> in <list>]
```

Example:

```
[3*x for x in [1, 2, 3, 4, 5]]
```

```
[3, 6, 9, 12, 15]
```

Functional Programming in Python

- filter(function, list)

```
1 def Positive(N):
2         if N > 0: return True
3         return False
```

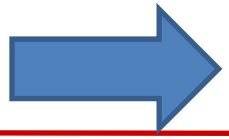
```
filter(Positive, [-10, 20, -2, 5, 6, 8, -3])
```

```
[20, 5, 6, 8]
```

- map(function, list)

```
1 def Mod4(N):
2         return N % 4
```

```
map(Mod4, range(1, 10))
```



```
[1, 2, 3, 0, 1, 2, 3, 0, 1]
```

Functional Programming in Python

- reduce(function, list)

```
1 def greater(A, B):
2         return A if A > B else B
```

```
reduce(greater, [1, 20, 2, -30])
```

```
20
```

Functional Programming in Python

METU Computer Engineering

Lambda Expression

- lambda *arguments* : *expression*
- Examples:

```
x = lambda a : a + 10
print x(5)
```

```
x = lambda a, b : a * b
print x(5, 6)
```

Exercises

- Write a function that calculates the factorial of a number without recursion or iteration.
- Write a function that calculates the average of the numbers in a list without recursion or iteration.