



Ceng 111 – Fall 2021

Week 10a

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.



Previously on CENG111!

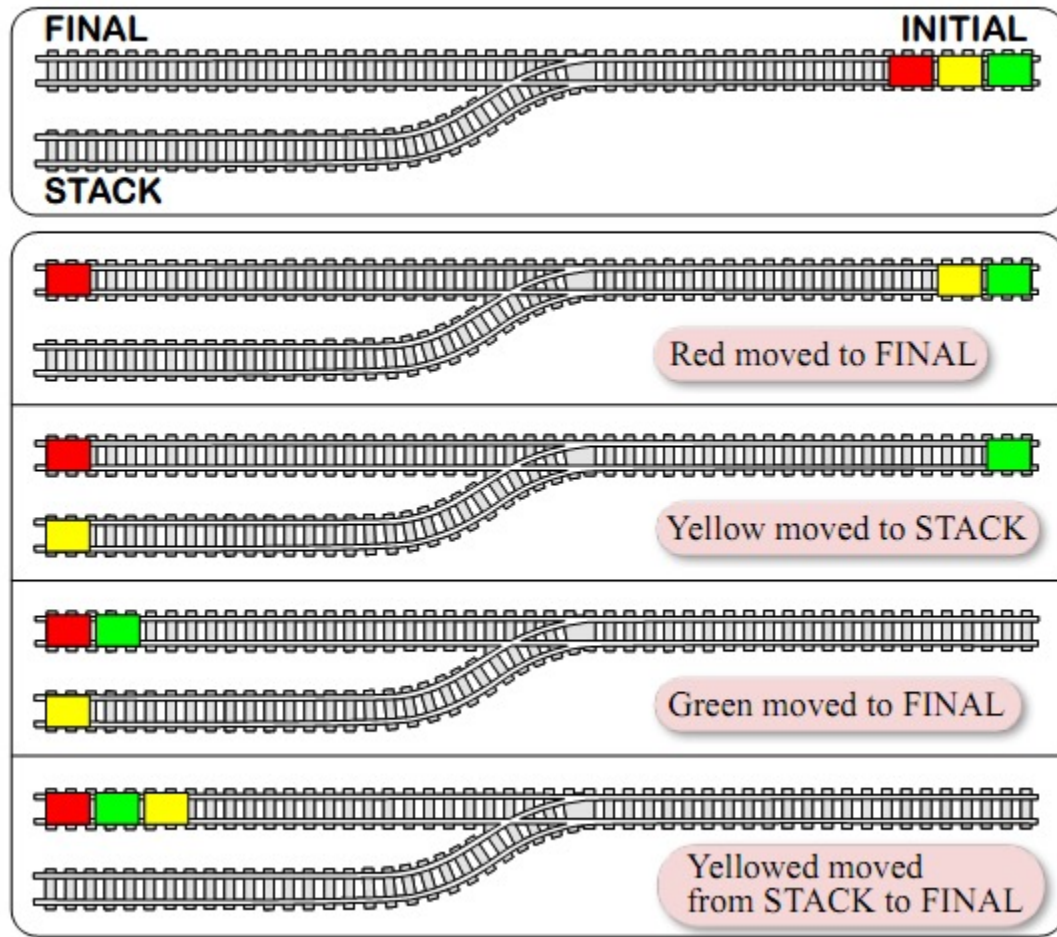
Expression Evaluation in PLs

- In most cases,
 - First, *Dijkstra's shunting-yard algorithm* is used to convert an expression into postfix notation.
 - Then, the postfix expression is evaluated using a *postfix evaluation algorithm*.



Previously on CENG111!

Dijkstra's Shunting-Yard Algorithm





Postfix Evaluation

1. Go from left to right
2. When you see an operator:
 - a) Apply it to the last two operands
 - b) Remove the last two operands and put the result in place of the operator.



Previously on CENG111!

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 <code>str()</code> function
r	Using the <code>repr()</code> 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.
```



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.



Previously on CEng111!

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
```

Parameter passing in functions

```

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

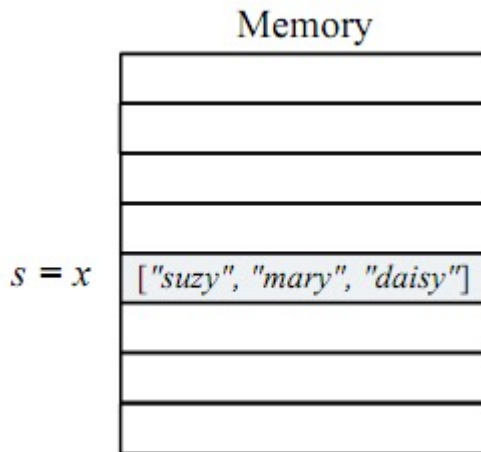
```

```

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

```

Call by Reference



(b)

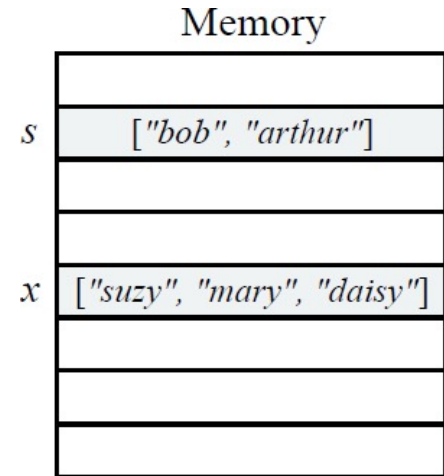
["suzy", "mary", "daisy"]
["suzy", "mary", "daisy"]

. Kalkan & G. Ucoluk - CEng 11

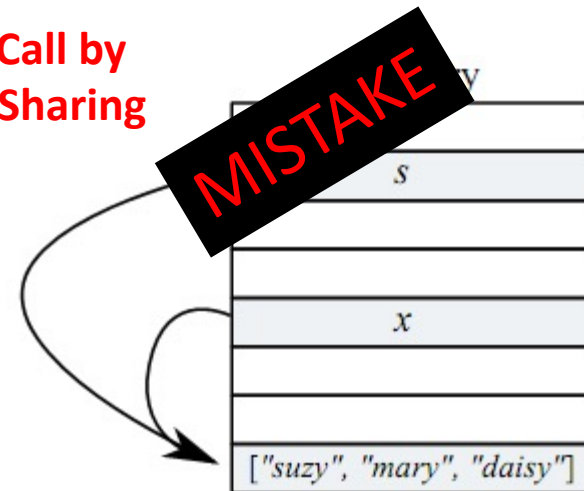
(a)

["suzy", "mary", "daisy"]
["bob", "arthur"]

Call by Value



Call by Sharing



(c)

["suzy", "mary", "daisy"]
["jennie", "arthur"]



Today

■ Functions



Administrative Notes

- THE2 announced:
 - Due date: 26 December, 23:59
- Midterm:
 - 22 December, Wednesday, 18:00



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
```



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]
```



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]
```




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

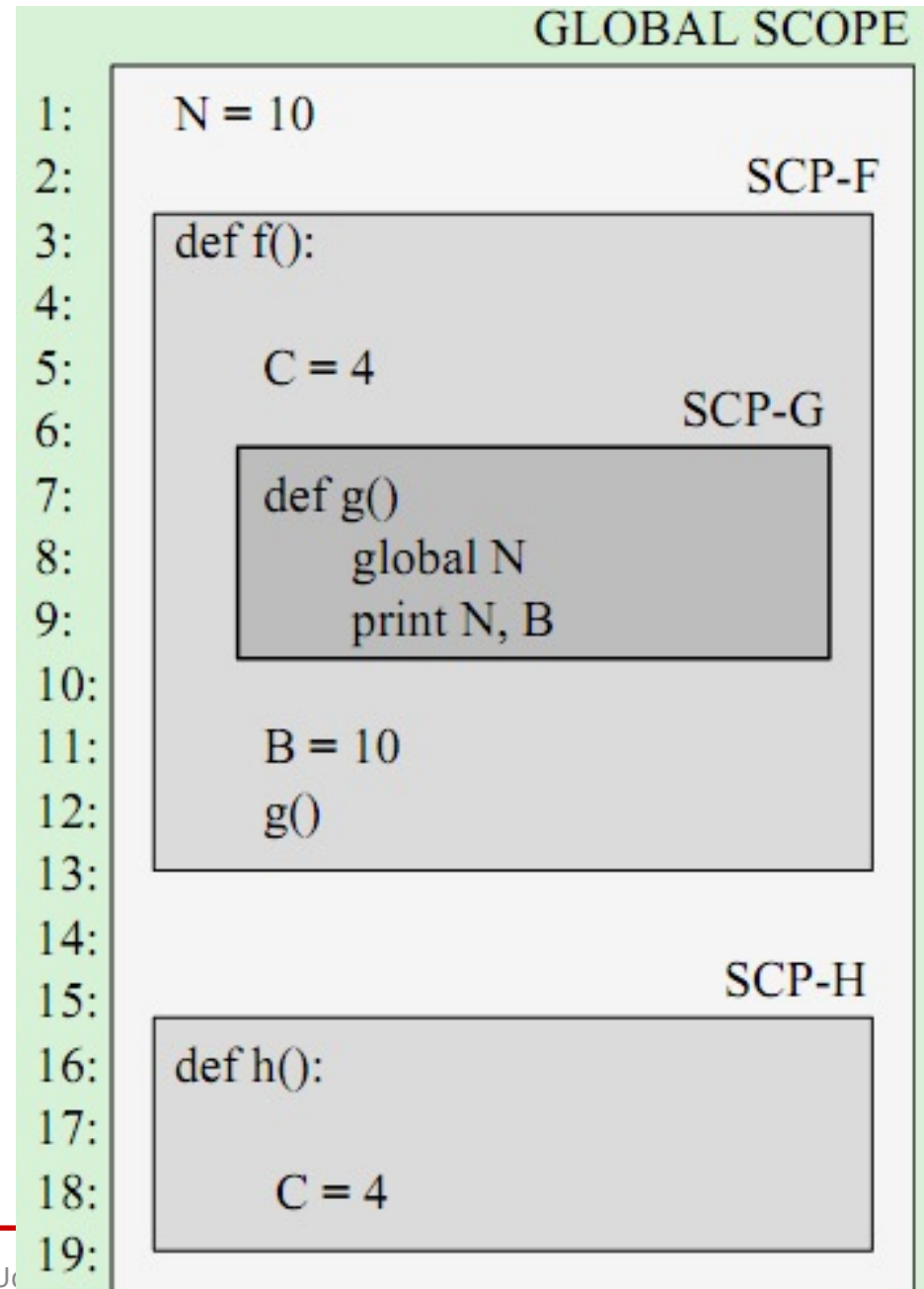
- To access variables in the global workspace, you should use “global <varname>”

```
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






■ Updating variables of an outer function


```
1 # Method using the function like an object.
2 def f():
3     f.a = 10
4     def m():
5         f.a = 20
6     m()
7     print("a (M1): ", f.a)
8
9 # Method using the nonlocal keyword (only with v3).
10 def g():
11     a = 10
12     def m():
13         nonlocal a
14         a = 20
15     m()
16     print("a (M2): ", a)
17
18 # Method using a mutable datatype
19 def h():
20     a = [1]
21     def m():
22         a[0] = 20
23     m()
24     print("a (M3): ", a[0])
25
26 # Call the functions
27 f()
28 g()
29 h()
```



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

■ 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

Lambda Expression

- *lambda arguments : expression*
- Examples:

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

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




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])`

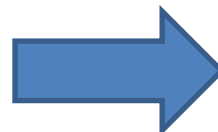


`[20, 5, 6, 8]`

■ 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



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**.