

More on Functions

IC152 Lecture 8
Feb 2021

More on Lists

```
9 # List of strings
10 lstStr = ['monday', 'tuesday', 'wednesday']
```

```
11
12 print(lstStr)
13 for x in lstStr:
14     print(x.capitalize())
15
```

```
17 # list of lists
18 lol = [[1,2,3],[4,5,6,4],[8,9]]
19
20 i = 0
21 for x in lol:
22     i = i+1
23     print('list',i)
24     for y in x:
25         print(y)
26
27 print('Individual element: ', lol[0][2])
```

```
list 1
1
2
3
list 2
4
5
6
4
list 3
8
9
Individual element: 3
```

Modularity of code

Python

```
# Main program

# Code to read file in
<statement>
<statement>
<statement>
<statement>

# Code to process file
<statement>
<statement>
<statement>
<statement>

# Code to write file out
<statement>
<statement>
<statement>
<statement>
```

```
def read_file():
    # Code to read file in
    <statement>
    <statement>
    <statement>
    <statement>

def process_file():
    # Code to process file
    <statement>
    <statement>
    <statement>
    <statement>

def write_file():
    # Code to write file out
    <statement>
    <statement>
    <statement>
    <statement>

# Main program
read_file()
process_file()
write_file()
```

This code
is more
modular

```

8
9 def avg3_v1(f,m,b):
10     foo = (f+m+b)/3
11     return foo
12
13 def avg3_v2(rd):
14     foo = (rd[0]+rd[1]+rd[2])/3
15     return foo
16
17
18 # main pgm
19
20 # get sensor readings
21 # lstReadings = getReaingsFromSensors()
22 lstReadings = [[13.2,9.4,3.5],[17.2,12.8,3.3]]
23
24 for room in lstReadings:
25     print('Average temp v1:', format(avg3_v1(room[0],room[1],room[2]),'.2f'))
26
27 print()
28
29 for room in lstReadings:
30     print('Average temp v2:', format(avg3_v2(room),'.2f'))
31

```

return statement

argument is a list

Functions can
return values back

Back to where?

From where the
function was called

```

Average temp v1: 8.70
Average temp v1: 11.10

Average temp v2: 8.70
Average temp v2: 11.10

```

formating for printing

```
9 import math
10
11 def SqRt(a):
12     """ A better sqrt function """
13     assert a >= 0, 'Negative input'
14     return math.sqrt(a)
15
16
17
18 x = float(input('Enter the number: '))
19 print('The sq root is:', SqRt(x))
20
21
```

docstring for documentation

assert statement: checks condition

```
Enter the number: 5
The sq root is: 2.23606797749979
```

```
Enter the number: -5
```

```
AssertionError: Negative input
```

```
In [55]: help(SqRt)
Help on function SqRt in module __main__:

SqRt(a)
    A better sqrt function
```

1h 11m
left



ALL



6. String reversal

Print a string in the reverse order using negative indexing and then reverse it again, but this time using positive indexing, to obtain the original string. Print both the strings in the same line after concatenating them.

Example: If your input string `s = "hello"`, then your output should be `"ollehello"`.

Hint:

```
s = "wonderful"
print(s[-4:-1]) #will print "rfu" and uses negative
indexing
print(s[5:2:-1]) #will print "red" and uses positive
indexing
```

Language: Python 3

Autocomplete Ready ⓘ

```
1 > #!/bin/python3 ...
10
11 #
12 # Complete the 'strRev' function below.
13 #
14 # The function accepts STRING s as parameter.
15 #
16
17 def strRev(s):
18     # Write your code here
19
20 if __name__ == '__main__':
21     s = input()
22
23     strRev(s)
24
```

Function does not
return anything

Non-editable

Functions: scope

Scope: region where you can unambiguously access a name

Scope: `global` or `local`

Ability to access a name depends on where it has been defined

in-scope or out-of-scope

LEGB rule for Python scope

LEGB is the order for name lookup

Local:

Names defined inside the block

Created at function call, not at definition

name is looked:

First in local

Then in enclosing

Then in global

Then in built-in

Not found, then error

Enclosing or nonlocal scope:

Only for nested functions

Scope is the enclosing function

Global scope:

Top-most scope

Built-in scope:

Special scope for built-in things: keywords, exceptions etc.

Local scope

```
8
9  def square(base):
10     result = base ** 2
11     print(f'The square of {base} is: {result}')
12
13  def cube(base):
14     result = base ** 3
15     print(f'The cube of {base} is: {result}')
16
17
18  x = 4
19  square(x)
20  cube(x)
21
```

```
The square of 4 is: 16
The cube of 4 is: 64
```

No confusion which
base is being
referred to

Enclosing or nonlocal scope

Only for **nested** functions: functions defined inside other functions

```
8
9  def outer():
10     var = 100
11     def inner():
12         print('Printing var from inner function',var)
13
14     inner()
15     print('Printing var from outer function:',var)
16
17
18
19
20  outer()
21
22
```

Local scope of `outer()` is
the enclosing scope of
`inner()`

```
10132/lectures/lec0_functions/code /
Printing var from inner function 100
Printing var from outer function: 100
```

Global scope

LEGB rule

Using global variables is considered **bad** programming practice

```
8  def f():
9      x = 5
10     g()
11     print('in f',x)
12
13  def g():
14      x = 7
15      h()
16
17  def h():
18      print('in h',x)
19
20
21  x = 3
22  f()
23  print('in main',x)
24
```

local scope

HW: what happens if this were a nested function?

global variable

```
in h 3
in f 5
in main 3
```

printing the global value of x

Summary of 3 scopes

LEGB rule

```
8
9  def outer():
10     # defines local scope of outer()
11     # also defines enclosing scope of inner()
12     def inner():
13         print(number)
14     inner()
15
16
17 number = 100
18 outer()
```

1. Inside `inner()`: local scope, but `number` doesn't exist
2. Inside `outer()`: enclosing scope, but `number` doesn't exist there either
3. Global scope, `number` exists here

100

→ Printing the global value
in this case

Built-in scope

LEGB rule

For built-in functions

eg. `len()`

Remember: do not redefine built-in names!