Lecture 6

Chapter 5

- 5.2 Functions
- 5.4 Case Study: computing the square root
- 5.5 Error Checking and Exceptions

5.2 Functions

Function definition:

```
def <name>(<formal parameters>):
    <body>
```

name – is an identifier
formal parameters – list (possibly empty) of variable names
body – is the body of the function,
all formal parameters are accessible only in this part.

Function invocation (function call):

```
<name>(<actual parameters>)
```

actual parameters – are the the ones whose values are assigned to formal parameters

5.2 Functions

optional parameter
with value by *default*

```
def countdown(start, end=1)
    for count in range(start,end-1,-1):
        print(count)
```

signature

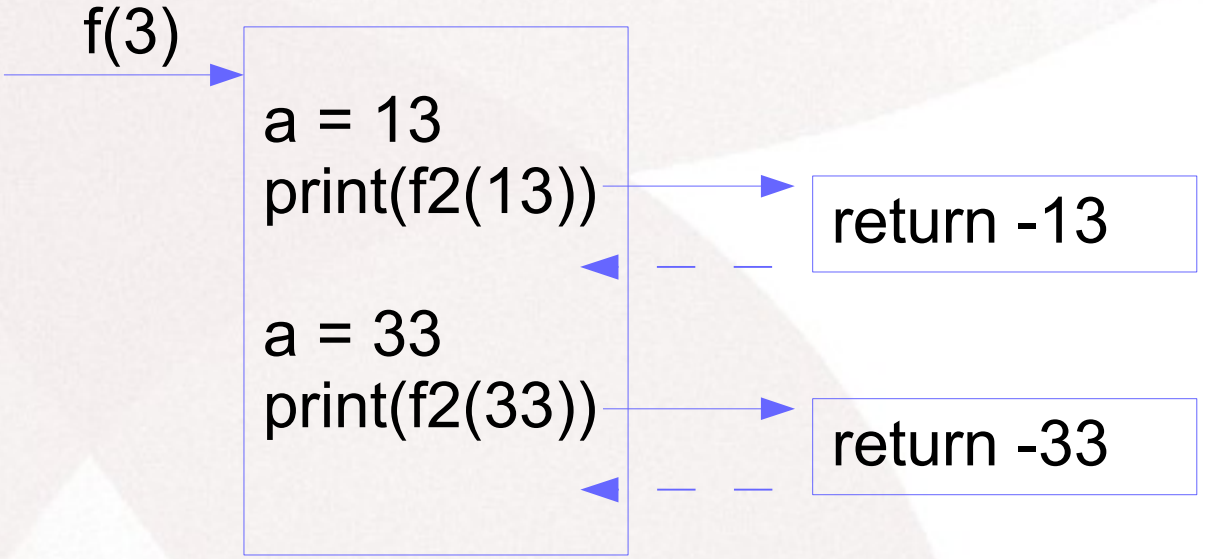
local scope

Convention: to avoid ambiguity, those parameters without default values must always be listed first in the signature.

5.2 Functions

```
def f1(a):
    a += 10
    print(f2(a))
    a += 20
    print(f2(b))

def f2(b):
    return b*(-1)
```



Each time a function is invoked, the system creates an internal structure 'activation record' to keep track of important information relevant to this particular execution of the function. It is maintained until the call is completed. See page 171 for more details.

5.2 Functions

Why do we need functions?

- mainly to make our programs easier to write, read, test, and to update/modify.

We can think of a function as a subprogram – a small program inside of a program.

We write a sequence of statements and give that sequence a name; then these instructions can be executed at any point in the program by referring to the function name.

5.2 Functions

- Functions may return more than one value
- Python *passes all parameters by value*
  (i.e. “original” values cannot be changed inside a function)
  exception: functions may modify variables that are mutable objects (*lists*, *graphics objects*)
- Other languages have a mechanism that allows to change parameter's values, that is called *call by reference*. Python doesn't have it.

5.2 Functions

Testing and debugging

Testing is an essential part of a program developing process. A program cannot be released unless it passes some testing.

An error in a program is called a bug.

The process of finding and eliminating the bugs is called debugging.

Two options for debugging:
- use the built-in debugger (if it is available in your IDE)
- use print statements

5.5 Error checking and Exceptions

Types of errors we met so far:

- NameError
- TypeError
- ValueError
- AttributeError

- belong to the `Exception` class.

Exceptions are used to report *cases that are out of ordinary*.

Usually, the result of an exception is to stop the execution of the code immediately.

For a programmer, the exception describes the immediate cause of a significant problem.

For a non-programmer, seeing such a reported error is not helpful (usually).

5.5 Error checking and Exceptions

Exceptions can be

- caught (try statement), and
- raised
  (raise <exception name>('phrase to be displayed'))

5.5 Error checking and Exceptions

Catching an exception (*try statement*):

```
try:
    body
except errorClass:
    recoveryBody
```

Example 1:

```
…
n = 0
while not 1 <= n <= 10:
   try:
      n=eval(input('Enter an integer from 1 to 10:'))
      if n<1 or n>10:
         print('Your number should be from 1 to 10')
   except ValueError:
      print('That is not a number.')
```

What does this program do?

see exceptionCatch.py

5.5 Error checking and Exceptions

Raising an exception:

Example 2:

```
…
n = 0
while not 1 <= n <= 10:
   n=input('Enter an integer number from 1 to 10:')

   if n.isdigit() == False:
      raise ValueError('it is not a (positive) number')

   n = int(n)
   if n<1 or n>10:
      raise ValueError('it is not in between 1 and 10,
including'
```

What does this program do?
see exceptionRaise.py, exceptionRaise2.py

5.5 Error checking and Exceptions

Example 3: Let's write a program that finds the sum of squares first *n* positive integers, where *n>1* and is given by the user.

$$\sum_{k=1}^{n} k^2 = 1^2+2^2+3^2+\ldots+(n-1)^2+n^2$$

$$\sum_{k=1}^{3} k^2 = 1^2+2^2+3^2 = 1+4+9+14$$

for *n=3*

Input: a positive integer

If a user inputs a non-number instance, the program should notify him/her about it, and prompt for the correct input. The user is given three chances to input the number correctly.

Output: a positive integer or notification of the error with input

Decision: Let's implement sum of square as a method

SumOfSquares

See the program sumOfSquares.py

5.5 Error checking and Exceptions

Example 4: Finaly let's recall our quadratic equations. $ax^2+bx+c = 0$, the quadratic formula

$$x_{1,2}=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$ gives us the solution right away, but

there are three cases for the expressions under the radical:

$D = b^2-4ac$ - determinant

when $D > 0$, there are two real-number solutions
when $D = 0$, there is only one real-number solution
when $D < 0$, there is no real-number solutions, moreover, if computer tries to extract a square root of a negative number, it will crash.

Let's write a program that will take into account all these cases.

see programs quadraticEquation.py, and quadraticEquationComplexRoots.py

In-Class Work

1. type in the following commands in Python shell and see what error messages you are getting (make sure you understand what generates the error):

```
>>>“abc”+23
>>> math.sqrt(-3)
```

```
>>> from math import sqrt
>>> sqrt(-9)
```

2. Show the signatures of each of the functions below, and draw the picture/figure of function invocations, after `f1(“Mary Jane”)` call.

```
def f1(s):
    n = st(s)
    n = ph(n)
    print(n)

def ph(s):
    return “Hello “ + s + “. How are you?”

f1(“Mary Jane”)
```

```
def st(s):
    ls = st.split()
    return ls[0]
```

In-Class Work (continues)

3.

```
def silly(x):
    print 'start silly'
    print x
    funny(2 * x)
    goofy(x - 1)
    print 'end silly'

def funny(y):
    print 'start funny'
    print y
    goofy(y + 1)
    print 'end funny'

def goofy(z):
    print 'start goofy'
    print z
    print 'end goofy'
```

Predict the output that results when silly(5) is invoked.

Homework Assignment

1. page 200 / 5.30

2. Write a program that finds a distance between two points (x_1,y_1) and (x_2,y_2), such that:

x_1, x_2, y_1, and y_2 are integers

$0 <= x_1,x_2 <= 30$, $0<= y_1,y_2 <=20$.

Do **type checking** (**TypeError**) and **value checking** (**Value Error**). You can raise errors or force the user to enter the correct values.