

Introduction into Python

Python 2: Programming, Control Structures, Functions

Daniel Polani

```
Code  
a = 3  
b = 4  
print a + b
```

```
7  
Code  
a, b = 3, 4  
if a > b:  
    print a + b  
else:  
    print a - b
```

-1

Programming II

```
Input  
>>> # Fibonacci series:  
... # the sum of two elements defines the next  
... a, b = 0, 1  
>>> while b < 10:  
...     print b  
...     a, b = b, a+b  
... 
```

1
1
2
3
5
8

Programming III

Features:

1. multiple assignment: rhs evaluated before anything on the left, and (in rhs) from left to right
2. while loop executes as long as condition is True (non-zero, not the empty string, not None)
3. block indentation must be the same for each line of block
4. need empty line in *interactive* mode to indicate end of block (not required in edited code)
5. use of `print`

Printing

```
Input _____
>>> i = 256*256
>>> print 'The value of i is', i
```

The value of i is 65536

```
Input _____
>>> a, b = 0, 1
>>> while b < 1000:
...     print b,      # comma prevents newline
...     a, b = b, a+b
... 
```

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

Flow Control

```
Input _____
>>> x = int(raw_input("Please enter an integer: "))
>>> if x < 0:
...     x = 0
...     print 'Negative changed to zero'
... elif x == 0:
...     print 'Zero'
... elif x == 1:
...     print 'Single'
... else:
...     print 'More'
... 
```

Note: use `elif` instead of `else if` to avoid indentation

Note: use as a switch statement

Iteration

Remark: Python `for` iterates over sequence (string, list, generated sequence)

```
Input _____
>>> # Measure some strings:
... a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print x, len(x)
... 
```

cat 3
window 6
defenestrate 12

Iteration II

Note: It is not safe to modify sequence while iterating over it. If necessary, create a copy:

```
Input _____
>>> for x in a[:]: # make a slice copy of the entire list
...     if len(x) > 6: a.insert(0, x)
...
>>> a
```

['defenestrate', 'cat', 'window', 'defenestrate']

Ranges

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> mylist
['foobar', 'foo', 'bar', ['in', 'ner', 'list'], 1.5]
>>> for i in range(len(mylist)): print i, mylist[i]
...
0 foobar
1 foo
2 bar
3 ['in', 'ner', 'list']
4 1.5
```

Daniel Polani: Introduction into Python – p.9??

Further Flow Control

- `break` ends the innermost `for` or `while` loop
- `continue` continues the next iteration of the loop, ignoring the remaining rest of the present iteration
- `else` as second clause of an iteration is performed when the loop ends regularly (not by a `break`)

```
Code
for n in range(2, 10): # notation as in program
    for x in range(2, n):
        if n % x == 0:
            print n, 'equals', x, '*', n/x
            break
    else:
        # loop fell through without finding a factor
        print n, 'is a prime number'
```

Daniel Polani: Introduction into Python – p.10??

The `pass` Statement

```
Code
while True:
    pass # the block must not be empty,
        # but if we do not want
        # to do anything, do pass
```

Function Definition

```
Code
def fib(n): # write Fibonacci series up to n
    """Print a Fibonacci series up to n.""" # help string
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b
```

```
Input
>>> # Now call the function we just defined:
... fib(2000)
```

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Function:

- definition by `def`
- name and argument list
- optional documentation string
- statements must be indented with respect to `def`

Daniel Polani: Introduction into Python – p.11??

Daniel Polani: Introduction into Python – p.12??

Functions

Properties:

- all variables are local, unless named `global` or system variables
- always a *call by reference*

Functions are Objects

```
>>> fib
```

```
<function fib at 10042ed0>
```

```
>>> f = fib
>>> f(100)
```

```
1 1 2 3 5 8 13 21 34 55 89
```

Note: functions are objects and can be assigned to variables

Note: `fib` is a procedure, i.e. a function that returns `None`.

Return Value: to return a value from a function, use the keyword `return`

Example

```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while b < n:
...         result.append(b) # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100) # call it
>>> f100 # write the result
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Note: `append` is a list method. It applies to the previous object.

Default Function Arguments

Example:

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'): return True
        if ok in ('n', 'no', 'nop', 'nope'): return False
        retries = retries - 1
        if retries < 0: raise IOError, 'refusenik user'
        print complaint
```

Use e.g. as:

```
ask_ok('Do you really want to quit?')
```

or

```
ask_ok("I am impatient, give me a quick reply", retries = 2)
```

Default Function Arguments II

Note:

- `in` is used outside of an iteration, as predicate identifying whether an object is in a list
- `raise` to raise exceptions

Note:

- call first the obligatory arguments, then the optional (even if optionals are named)
- only one value assignment per argument