# Basic Python
for Jeppesen Products

Course manual

Covers Python version 2.6
used in Jeppesen Products

JEPPESEN®
A BOEING COMPANY

Table of contents

General

Python is a platform independent programming language that combines the simplicity of script languages like Perl and Awk with the power of a real programming language, such as C++ or Java. Python is Open Source and is freely available for both commercial and non-commercial use.

Python is one of the three programming languages you have to be familiar with as a CARMUSR developer. The other two are Rave and PDL.

In Jeppesen Products, Python is used for various tasks when you customize the system. Examples of such uses include:

- ***Stand-alone programs***. You could use Python to implement programs that are started from Studio, e.g. for conversion of data from other systems.
- ***Planning Commands***. Studio has a rich Python API that makes it possible to create your own (problem specific) GUI commands. It is actually possible to automate large parts of the planning process using Python.
- Create ***Report definitions*** using PRT (Python Reporting Toolkit).
- ***Rostering optimization***. Matador has a Python API which makes it possible define how the optimization should work, down to the smallest detail.

In addition to the component specific APIs, there are some generic Python modules delivered with the CARMSYS that can be used in all cases (stand-alone programs, Studio and Matador)

Please note that *this course has its focus on the Python language itself* and on how to develop Python applications in a Linux/Unix environment.

PRT, the Studio-Python-API, the Jeppesen-Generic-Python-Modules and the Matador-Python-API and related APIs are covered in other courses.

Slides

Exercises

Introduction

Purpose

Here you can for each exercise read what is practised in it.

Log in

To login to the Academy network, take a computer station and double-click the blue icon on the top left. You are greeted with a Windows desktop.

Look in the left upper corner of the monitor to the left. You will see a label “studentNN” where NN is a number. That will be your username in the academy network.

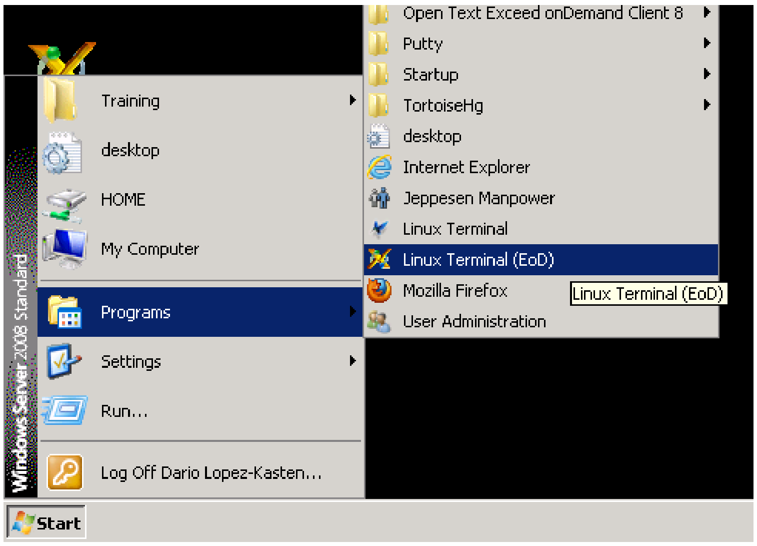
The password is “password”

Getting started

Start a terminal

All exercises will be executed in a terminal.

1. To start a terminal, got to the **Start → Programs** menu and Choose "Linux Terminal (EoD)"



Exceed on Demand will be started, and will prompt for your username and password again. After logging in, an Xterm window will open.

2. Once Exceed has started, start the Terminal application from the Windows Start Menu: ***Start → Programs → Linux Terminal***

Text Editor to use

To write your programs, you can either use a terminal based editor or a Windows based editor. In the Academy network, both Emacs and Vi are available for terminal editing.

For a Windows based Editor, you can use ***Notepad*** ++. It is a fairly easy to use TextEditor that provides Python Syntax Higlighting. You can find it at this location on your windows Computer:

Start → Programs → Notepad++ ***→ Notepad***++

Ask your teacher if you have any questions.

Exercise 1

Purpose

We will also become familiar with the different ways to run Python programs.

Instructions

We have not learned much about the Python language yet.

NOTE! Use this Python source code in the first exercise.

```
a = 10
b = 20
print "Result a+b:", a + b
```

The `print` statement prints values to the terminal.

Step 1 Default Python version

1. Start a terminal
2. What Python version do you get if you start the interpreter with the command `python`? (Type this command in the terminal).
3. Exit the python interpreter, by typing `exit` or `quit`.
4. What happens?
5. To exit the Python interpreter, follow the instructions in the terminal window.
6. Now, try typing the following commands in the terminal
    - `python2.6`
    - `python2.7`
    - `python3.5`
7. What results do you get?

Step 2 Get Python 2.6 from our software

- Start Studio

  ```
  cd
  cd PYTHON1_system
  ./PYTHON1_in_linux_start.sh
  ```

  (ignore the warning about not finding default RAVE program)
- Execute the menu entry `Special -> Start xterm`.

In the (red and ugly) new window, give the commands

```
cd

cd PYTHON1_course
xterm -title "Python Course"
```

- Now run the command `python`.
- What version do you get?

Step 3 Start the interpreter from the command line.
Type the source code specified in the *Instructions* section above manually, then exit the interpreter.

> **Hint:** `Ctrl-p` (or ↑) gives you access to rows you have entered in the interpreter. `Ctrl-n` (or ↓) takes you back again.

Step 4 In the terminal, call the interpreter with the `-c` flag and specify the above code in the call.
Remember that each python command has to be on a line of its own.

Step 5 Write the source code into a file and give the filename as argument to the interpreter. (Put the file in the directory `$HOME/PYTHON1_course`.)

Step 6 Start the interpreter again and execute the code in your file using the built-in-function `execfile`.

Step 7 Put the source code in a Unix script and verify that it works.

> **Hint:** Don’t forget to make the file executable.

Step 8 Use the **help** function.
Start the interpreter and type:

```
help()
```

This is how you get interactive help in the interpreter.

```
help(<object>)
```

This is how you get help about a specific object (in this case the “`dir`” python command)

```
help(dir)
```

`dir(<object>)` is often useful. It gives you a list of all the attributes of an object.

Built-in-data-types

Exercise 2

Purpose

Become familiar with built-in-data-types in Python.
Start to use the course book.

Instructions

Use the interpreter and try the following expressions. Explain why you get the results you get. Use `help()`, the course book and the slides to find the answers.

Exercise 2.1 Numbers

1. `2 ** 63, sys.maxint`
2. `2/5*10, int(2/5.0*10), int(2/5*10.0), 2*10/5`
3. `1j * 1j, (1j * 1j).real`
4. `math.sin(math.pi), round(math.sin(math.pi), 15)`

Exercise 2.2 Strings

1. `s1 = "Pi =" + str(math.pi)`
2. `s1 * 2`
3. `s1[5:]`
4. `"Pi = %.6f" % math.pi`

Exercise 2.3 Tuples

```
t = ("Ten", 20, 30, 40, 50, 60)
```

1. `t[0], t[-1]`
2. `t[:], t[:-1]`
3. `len(t+t)`
4. `4 in t, 40 in t`
5. `t[44:]`

Exercise 2.4 Lists

```
L1 = [10, 20, 30]
L2 = L1
```

1. `L2.reverse() ; L1`
2. `L1.sort(); L2`

3. `L2.append(99); L1`
4. `L2.index(20)`
5. `L2 = (L1 * 2)[2:-2]; L1, L2`
6. `L2[-1:] = []; L2`

Exercise 2.5 Dictionaries

1. `d={(0,0): "X", (0,1): "Y", (1,0): "A"}; d[(0,1)]`
2. `del d[(0, 0)]; d.items()`
3. `d[(1, 1)] = "X"; d.keys()`
4. `d.has_key((1, 1)), len(d)`
5. `d.update([(1, 2), (3, 4)]); d`

Exercise 2.6 Sets

```
s1=set([1, 2, 3, 4]); s2=set([3, 4, 5, 6])
s1 & s2; s1 | s2; s1 ^ s2; s1 - s2
s1.update([7,8]); s1
s1 -= s2; s1
```

Exercise 2.7 Boolean

```
a = [2, 3]; b = a[:]
b == a; b is a; a or b; (a or b) is a
True * 35
```

Exercise 2.8 Files

Write this code in a separate file, and run it (substitute ***SID*** below with your student id):

```
f = open("/users/SID/PYTHON1_system/PYTHON1.cfg")
lines = f.readlines()
print "’%s’ has %s rows." % (f.name, len(lines))
f.close()
```

Exercise 2.9 Misc [EXTRA]

a) The `[ ]` operator for strings:

```
    s = "H"; s[0][0][0][0], s==s[0]
```

Why do you get this result?

b) Immutable object

```
a = "Hello"
a[1:3] = []
```

Why do you get an error?
What to do instead?

c) ?

In the programming language C you could do:

```
a ? b : c
```

meaning "if a then b else c".

Find an **expression** in Python doing the same.
The execution must be lazy. Only the `b` **or** `c` should be executed.

You could verify that it is lazy by letting `b` and `c` be the following expressions:

```
  sys.stdout.write("b is here\n")
```

and

```
 sys.stdout.write("c is here\n")
```

d) Lists - More about references and copies. [EXTRA EXTRA]

Consider the following code:

```
import copy
s = "Hi"; LS = list(s); L = [[LS] * 2]*2; L
L1= copy.deepcopy(L)
L2= copy.copy(L)
L3 = L[:]
L4 = L
L3[0][0][-1:] = []
del L3[0]
del L3[0][1]
L4[1] = "Hi"
L1[1][1][1] = "o"
```

After execution - what do `L1, L2, L3` and `L4` contain and why?
Draw a picture!

Program Flow

Exercise 3

Purpose

Get familiar with the statements for program flow in Python.

Exercise 3.1 Manipulate the objects in a list

You should now implement code that increase even integers in an integer list with 10, in different ways.

You should update the list (not create a new one).

Test data:

```
l = m = [2,4,5,6,7,7,8]
```

Apply your code on `l` and make sure that `m` has been changed.

```
>>> m
[12, 14, 5, 16, 7, 7, 18]
```

1. Use a **`while`** loop
2. Use a **`for`** loop instead
3. Use a list comprehension

Hint: Look up functions `range`, and `enumerate` in the course book; they may be useful.

Exercise 3.2 Last visited station

The file `$HOME/PYTHON1_course/Large.etab` contains information about when crew members last visited airports. Create a dictionary with **crew number** and **airport** as key and the **date for the visit** as data.

Then use the dictionary to find out when crew 721111 visited NRT and when crew 742246 visited MIA.

Try to make a solution that needs less than 4-5 seconds to execute.

Hint: The external table is very large. It could be a good idea to create a smaller version of the file and use that that one while trying out your code.

Exercise 3.3 Reduce size of external table [EXTRA]

This is a part of some data converting we had to do for a benchmark in the beginning of 2007.

Create a new external table, `named crew_qualifications_2.etab`, from `crew_qualifications.etab`. Only data about crew based in **DFW** or **LAX** should be available in the new table. You find information about the home base for each crew member in `crew_attributes.etab`.

Hint: The number of digits in the **`crew`** column varies.

Hint: Look for useful string methods on page 41-44.

Functions

Exercise 4

Purpose

Get familiar with functions in Python.

Exercise 4.1

Your first function

Step 1 Create a function `my_add` that takes any number of (not named) arguments and returns the sum of them.

Hint: Use the “+” operator.

Test it with the following calls:

```
>>> my_add("Jeppesen ", "Systems ", "AB")
'Jeppesen Systems AB'
>>> my_add([1,2],[4,5],[],[45])
[1, 2, 4, 5, 45]
>>> my_add(1, 2, 3, 4, 5.0)
15.0
>>> my_add()
None
>>> my_add(0)
0
```

Step 2 Use the `reduce` function to implement your function.

Hint: Read about the `reduce` function in the course book.

Step 3 [EXTRA] Use recursion.

Exercise 4.2

Manipulate the objects in a list

Now you have more possibilities for solving the list manipulation problem in 3.1.

You should update the list (not create a new one).

Test data:

```
l = m = [2,4,5,6,7,7,8]
```

Apply your code on `l` and make sure that `m` has been changed.

```
>>> m
[12,14,5,16,7,7,18]
```

1. Use the `map` function?
2. [EXTRA] Could you use recursion?

Which of the solutions do you prefer?

Exercise 4.3 Lambda expression

Step 1 **Filter**
Use the `filter` function and remove all items larger than 10 from the list `l`.
Implement the same using list comprehension.
Which solution do you prefer?

Step 2 **Sort**
Sort the list `l` (in place!) to get descending order.

Hint: The course book describes list method `sort` on p. 41, and the built-in functions `cmp`, p. 203 and `sorted`, p. 211, which may prove useful.

Exercise 4.4 Sort [EXTRA]

Sort one list using the order of another:

```
a = ['a','b','c','d','e']
b = [4,3,6,1,9]
print mySort(a,b)
['d', 'b', 'a', 'c', 'e']
```

Hint: `dict`, `zip` and `lambda` could be useful

Modules

Exercise 5

Purpose

Get familiar with modules in Python

Preparations

Read through pages 143-144 in the course book.

Step 1 Basics

In the very first exercise you created a file with the following content.

```
a = 10
b = 20
print "Result a+b:", a+b
```

Create and `import` a file with that content.

Change the value of the variable `a` in the file.
Import the file again. Has the value of `a` changed?

Do `reload` and check the value of `a`.

Exit from the interpreter.

Remove the source code file.
Start the interpreter again. Import your module.
Why does it work?

Step 2 Create module (`mod1.py`) that:

- Defines a function (`func`) that returns the number of times it (the function) has been called.
- Contains a documentation string. (See page 30 in the course book.)
- Writes the message "`Do not use mod1.py as script`" if used as script.

Hint: `sys.argv[0]` gives you the name of the executed file.

Hint: Run your program from a shell prompt to verify that you get the message.

Step 3 Create another python file (`mod1test.py`) that imports your module, calls the function 5 times and then prints a string containing the returned value. Run that program from the shell prompt.

Step 4 Start the interpreter.

`Import mod1`. Verify that you don't get a message.

Look at the values of:

```
>>> help(mod1)
>>> print mod1.func.__doc__
>>> dir(mod1)
```

Step 5 `from` and `reload`

Restart the interpreter and do the following.
Explain why you get what you get:

```
import mod1
from mod1 import func as ff
ff(); mod1.func(); mod1.func is ff
reload(mod1)
ff(); mod1.func(); mod1.func is ff
```

Step 6 Update the file `mod1.py` (without leaving the interpreter).

Change the name of the variable containing the counter and then try.

```
reload(mod1);ff();mod1.func()
```

Why do you get that result?

Classes

Purpose

To get some experience from classes and object oriented programming in Python.

Exercise 6

Exercise 6.1 A stack

Lets build a stack data structure in different steps.

Step 1 Create a class called **`Stack`** that offers the methods `pop()` and `push(item)`. (a new stack should always be empty)

Test:
```
>>> s = Stack()
>>> s.push(1); s.push(2)
>>> s.pop()
2
>>> s.pop()
1
```

Step 2 Define __`repr`__ and __`str`__ methods, so that printing a stack shows its content.

Test:
```
>>> s = Stack()
>>> s.push(1); s.push(2)
>>> s
S[1,2]
>>> print s
S[1,2]
```

Step 3 Define a __`len`__ method to let `len` return the current number of objects in the stack.

Test:
```
>>> s = Stack()
>>> s.push(1);s.push(2)
>>> len(s)
2
```

Step 4 Let the operation "`s += item`" be the same as the "`s.push(item)`" method call. Read pages 60-61 in the course book to see what operators map to which special methods.

Test:
```
>>> s = Stack(); s+=1; s+=2; print s
S[1,2]
```

Step 5 Instance counter
Add a variable `num` to the `Stack` class, to keep track of the number of created (still living) instances `Stack`. Provide a class method, `num_instances`, that returns the number of instances of the class.

Hint: Add a `__del__` method and change the `__init__` method.

Test:

```
>>> s = Stack(); Stack.num_instances()
1
```

Exercise 6.2 Last visited station – again [EXTRA]

Improve the implementation you made for Exercise 3.2, to use a class.

Exercise 6.3 Module Magic [EXTRA] [EXTRA]

Purpose

Deeper understanding how Python accesses attributes and how modules are managed.

Preparations

In order to be able to carry out this exercise, you need to understand how the module attribute `__name__` works, understand the contents of `sys.modules` and understand how `hasattr` works.

Use the course book for reference.

Create a module that keeps track of how many times it has been reloaded.

When you import and reload the module from the python interpreter, an output similar to this should be seen:

```
>>> import testmod
Initial import of <module 'testmod' from 'testmod.py'>
>>> reload(testmod)
<module  'testmod'  from  'testmod.pyc'>  has  been  reloaded  1
time(s).
<module 'testmod' from 'testmod.pyc'>
>>> reload(testmod)
<module  'testmod'  from  'testmod.pyc'>  has  been  reloaded  2
time(s).
<module 'testmod' from 'testmod.pyc'>
...
>>> reload(testmod)
<module  'testmod'  from  'testmod.pyc'>  has  been  reloaded  20
time(s).
<module 'testmod' from 'testmod.pyc'>
>>> print testmod.reload_counter
20
>>>
```

Exception Handling

Exercise 7

Purpose

To get some experience from exception handling in Python

Exercise 7.1 Calculator

Write a simple calculator that takes (mathematical) expressions and evaluates them. It should handle exceptions, and end when the input string is empty.

```
>>> my_calc()
Expression: 10 / 2
5
Expression: 5 + strange
Bad input '5 + strange' : name 'strange' is not defined
Expression:
>>>
```

Hint: Look up and read about the functions `raw_input()` and `eval()`.

Exercise 7.2 The Stack

Update your 'Stack' class from Exercise 6 with exception handling. Raise an empty-stack-exception when trying to `pop()` an empty stack and full-stack-exception when there are **too** many items in the stack. You need to define new Exception classes. They should be sub classes to `IndexError`.

```
>>> s.pop()
Traceback (most recent call last):
File "<stdin>", line 1, in ?
File "<stdin>", line 11, in pop
__main__.EmptyStackError: No more items
>>>
```

```
>>> s.push(1)
Traceback (most recent call last):
File "<stdin>", line 1, in ?
File "<stdin>", line 7, in push
__main__.FullStackError: The stack is full
>>>
```

Iterators & generators

Exercise 8

Exercise 8.1 Create generators

The purpose of this exercise is to learn more about iterators and generators. We will create a callable object `myGen` that returns a generator. The generator should provide the values `x**x` for `x` in `1,2,3,4,5...` as long as `x**x < sys.maxint` is true.

Use these statements to verify that your generator object works as intended:

```
for a in myGen():
   print a,
```

and

```
sum(myGen())
```

Step 1 Create a generator function. (Use the `yield` statement)

Step 2 Use existing generator functions in the `itertools` module.
(You should not use any `yield` statement.)

Hint: read up on `itertools.takewhile, itertools.imap` and `itertools.count`

Step 3 Make a solution without using `itertools` or the `yield` statement.

Hint: Create a class

Profiling, Testing & Built-in Modules

Exercise 9

Exercise 9.1 Weak references

Step 1 Read about the `weakref` module in the course book.

Step 2 Use the Stack class you created earlier.

1. Create two Stacks, `s1` and `s2`.
2. Push `s1` into `s2` and `s2` into `s1`.
3. Delete `s1` and `s2`.
4. Check the number of `Stacks` that still are alive.

Step 3 Could a weak reference solve the problem?

Do the same as in Step 2, but push a weak reference to `s1` into `s2`.
Are the Stacks killed now?

Note Garbage collection (gc) **does** detect and remove “cyclic garbage”, but not that frequently (you could read about the `gc` module).
However, when you use Cfh in Studio you must avoid cyclic references.
Gc is not able to detect them and you get very magic behaviour if you create more than one form with the same name.

Exercise 9.2 subprocess

Read about the module in the course book.

Then copy and study the script:
`/carm/academy/NiceToHaveIQ/lib/python/nth/examples/unix_call.py`

Try the script with different shell commands.

Exercise 9.3 Performance

Sometimes it matters.

Use the `timeit` module to compare the following methods to create a long list filled with 0.

Step 1

```
l=[]
for dum in range(10000): l.insert(0,0)
```

Step 2

```
l=[]
for dum in range(10000): l[0:0]=[0]
```

Step 3
```
l=[]
for dum in xrange(10000): l[0:0]=[0]
```

Step 4
```
for dum in xrange(10000): l.append(0)
```

Step 5
```
[0 for dum in xrange(10000)]"
```

Step 6
```
import itertools as it
[0 for dum in it.repeat(None,10000)]
```

Step 7
```
[0]*10000
```

Do you see any significant differences?

Could you explain them?

Exercise 9.4 Doctests

We will revisit exercise 6.1 and create doctests for it.

Step 1 Study the instructions for exercise 6.1 and figure out a way to design your tests

Step 2 We start by creating a doctest. For the sake of simplicity, we will create our tests in the same directory as the source file for exercise 6.1.

1. Create a file `test_ex61.txt` that will contain all of your doctest text.
2. create a file `test_ex61.py` that will contain your test runner.

Hint: Look at the slides for tips on how to go about creating doctests!

Step 3 Run your tests and see if they pass

Step 4 Add the ability to concatenate two stacks. Items in the new stack should be in the same order as the stacks where added together.

Step 5 Add a test for this new ability.

Exercise 9.5 Unittests

Do the same as in previous exercise but with unittests instead of doctests.

Hint: Look at the slides on unittests to see how to create unittests!

Was it harder or easier?

Which do you prefer?

Which would go faster for you to work with in parallel with development?

Exercise 9.6 Print the largest files [EXTRA]

The following `shell` command prints a sorted list with size and relative path to the 10 largest files in a directory tree:

```
> find . -type f -printf "%10s %p\n" | sort -n -r |
head -10
```

Write a python script that does the same.
The script should support 4 flags.

`-n NUMBER_OF_FILES`	: default, 10
`-d TOP_DIRECTORY`	: default, current directory
`-o FILE_OWNER`	: default, any
`-h`	: Show some help

You could try to make 2 solutions. One that use the find command above and one that does not use any Unix commands.

Hint: `os.walk()` and `os.stat()` could be useful.

Hint: You need the `pwd` module to get the UID given the user name.

Hint: Use `getopt` for the parameter handling.

Exercise 9.7 Profiler [EXTRA]

Add support for profiling your two solutions from the previous exercise.
Let the script you made also support the flags:

`-p` : use profiler, default is False.

`-2` : use the second implementation, default is the first implementation.

Use a large directory tree when testing. E.g. "`/opt/Carmen/Systems`" if available.

Solutions

Exercise 1 Get started

Step 1 Default Python version

When you type `python` in a terminal, you will always get the default system version (unless you have modified your environment).

Red Hat uses python for its RHEL server administration software, and Python 2.6 is the default Python on RHEL6.

Jeppesen products are verified on Python 2.6.6

Exiting the python interpreter is a bit backwards. The interpreter is smart enough to understand that you are trying to exit, but not smart enough to actually quit. You have to press the `Ctrl` key while simultaneously pressing the `D` key.

Typing `python2.x` is a way to explicitly say which python version you want to use. In this case we want to specify different versions of the Python 2 series. Not all released Python versions are installed or available in any given system.

Step 2 Get Python 2.6

You still get python2.6.

Why? You started Studio and current Jeppesen products are using Python version 2.6.

So what is the difference of starting Python inside and outside of Studio?

When you start Studio a couple environment settings are prepared for you:

1. `CARMUTIL/bin` is added to `PATH` when Studio is started.
2. `CARMUTIL/bin/python` is a link to `/usr/bin/python2.6`.

Last but not least, when Studio is running, it makes it’s Python APIs available to you. Without these you cannot program with Studio.

More on this in the *Python in Studio* course.

Note It is fine to use **`python2.6`** as the command to invoke python, instead of using **`python`** during the course. However, you should (of course) NEVER specify anything but **`python`** in a CARMUSR, to make sure that the CARMUSR uses the Python version that the CARMSYS prefers.

In the solutions below we will use "python" without any reference to a specific version.

Step 3 Interactively

```
> python
>>> Type the source code.
>>> CTRL-D
>
```

Step 4 `-c flag`

```
> python -c "a = 10 \
? b = 20 \
? print a+b"
30
```

Step 5 With a file as argument
Create the file `$HOME/PYTHON1_course/Ex1`
Run:

```
> cd $HOME/PYTHON1_course
> python Ex1
```

Step 6 Use `execfile`

```
> python
>>> execfile("Ex1")
>>> CTRL-D
>
```

Step 7 Run as Shell script
Add as first row to `Ex1`:

```
#!/usr/bin/env python

> chmod u+x Ex1
> ./Ex1
```

Step 8 `---`

Exercise 2 Built-in-data-types

Exercise 2.1 Numbers

1.
```
>>> import sys
>>> 2 ** 63
9223372036854775808L
>>> sys.maxint
922337203685477580
```

The result is larger `than MAXINT (2^63-1)`. The type is changed to a Long Integer. The module `sys` contains the definition of `maxint`.

2.
```
>>> 2/5*10, int(2/5.0*10),int(2/5*10.0), 2*10/5
(0, 4, 0, 4)
```

"/" and "*" have the same precedence in expressions. The calculation is done from left to right (see Chapter 4, pages 65-66) "/" means true integer division when applied on integers. The operands in a (sub) expression are changed to the most complex type before the calculation takes place.

```
3.>>> 1j * 1j, (1j * 1j).real
   ((-1+0j), -1.0)
```

The attribute `real` returns the real part of a complex number object.

```
4.>>> import math
  >>> math.sin(math.pi), round(math.sin(math.pi),15)
  (1.2246063538223773e-16, 0.0)
```

There are often small rounding errors when you use `float`. `round()` is a built in function.

Exercise 2.2 Strings

```
>>> s1 = "Pi = " + str(math.pi)
>>> s1 * 2
'Pi = 3.14159265359Pi = 3.14159265359'
>>> s1[5:]
'3.14159265359'
>>> "Pi = %.6f" % math.pi
'Pi = 3.141593' #rounded
```

`str` is the name of the string data type.

Exercise 2.3 Tuples

```
>>> t = ("Ten",20,30,40,50,60)
```

```
1.>>> t[0], t[-1]
   ('Ten', 60)
```

```
2.>>> t[:], t[:-1]
   (('Ten', 20, 30, 40, 50, 60), ('Ten', 20, 30, 40, 50))
```

```
3.>>> len(t+t)
  12
```

`len` is a built in function.

```
4.>>> 4 in t, 40 in t
   (False, True)
```

```
5.>>> t[44:]
   ()
```

There is no check that you slice out existing items from a sequence.

Exercise 2.4 Lists

```
>>> l1 = [10,20,30]
>>> l2 = l1
```

Note that `l1` and `l2` are references to the same object.

```
1.>>> l2.reverse() ; l1
   [30, 20, 10]
```

"`;`" can be used to define a sequence of python statements on one row.

2.
```
>>> l1.sort(); l2
[10, 20, 30]
```

3.
```
>>> l2.append(99); l1
[10, 20, 30, 99]
```

4.
```
>>> l2.index(20)
1
```

The `index` method returns the position of the first occurrence of the argument found in the list. It is available for all sequence objects.

5.
```
>>> l2 = (l1 * 2)[2:-2]; l1,l2
([10, 20, 30, 99], [30, 99, 10, 20])
```

Both the `*` operator and the right hand slicing creates a new list. `l1` and `l2` are references to different objects after the operation.

6.
```
>>> l2[-1:] = []; l2
[30, 99, 10]
```

The last item in the list has been removed.

Exercise 2.5 Dictionaries

1.
```
>>> d={(0,0):"X", (0,1):"Y", (1,0):"A"}; d[(0,1)]
'Y'
```

2.
```
>>> del d[(0,0)]; d.items()
[((1, 0), 'A'), ((0, 1), 'Y')]
```

3.
```
>>> d[(1,1)] = "X"; d.keys()
[(1, 1), (1, 0), (0, 1)]
```

4.
```
>>> d.has_key((1,1)), len(d)
(True, 3)
```

5.
```
>>> d.update([(1,2),(3,4)]); d
{(0, 1): 'Y', 1: 2, (1, 0): 'A', 3: 4, (1, 1): 'X'}
```

Exercise 2.6 Sets

```
>>> s1=set([1,2,3,4]); s2=set([3,4,5,6])
>>> s1&s2; s1|s2; s1^s2; s1-s2
set([3, 4])
set([1, 2, 3, 4, 5, 6])
set([1, 2, 5, 6])
set([1, 2])
>>> s1.update([7,8]); s1
set([1, 2, 3, 4, 7, 8])
>>> s1-=s2; s1
set([1, 2, 7, 8])
```

Exercise 2.7 Boolean

```
>>> a = [2,3]; b = a[:]
>>> b == a; b is a; a or b; (a or b) is a
True
False
[2, 3]
True

[2,3] is not empty --> True and a or b returns a if a is true.
```

```
>>> False * 35; True * 35
0
35
```

`True/False` is converted to `1/0` in arithmetic expressions.

Exercise 2.8 Files

```
>>> f = open("/users/SID/PYTHON1_system/PYTHON1.cfg")
>>> lines = f.readlines();
>>> print "'%s' has %s rows." % (f.name, len(lines))
'/users/SID/PYTHON1_system/PYTHON1.cfg' has 14 rows.
```

Note that the second argument to the string format operator (`%`) must be a tuple.

```
>>> f.close()
```

You don't have to close the file. That is done automatically when all references to the file object are gone. However, it is a good habit to do it.

Exercise 2.9 Misc

a) The `[]` operator for strings.

```
>>> s = "H";s[0][0][0][0],s==s[0]
('H', True)
```

`s[0]` returns a string containing the first character in `s`.
That means that `s[0] == s`, if `s` only has one character.

b) Immutable objects

```
>>> a = "Hello"
>>> a[1:3] = []
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in ?
TypeError: object doesn't support slice assignment
```

Immutable objects do not support operations that change objects.

`a = a[0] + a[3:]`
does what we want. A new string object replaces the original one.

c) ?

A general solution:

```
 (a and [b] or [c])[0]
```

If you know that `b` is `True` the following is enough:

```
a and b or c
```

d) Lists - references and copies

```
>>> import copy
>>> s = "Hi";l = list(s); L = [[l] * 2] * 2;L
[[['H', 'i'], ['H', 'i']], [['H', 'i'], ['H', 'i']]]
>>> L1= copy.deepcopy(L)
```

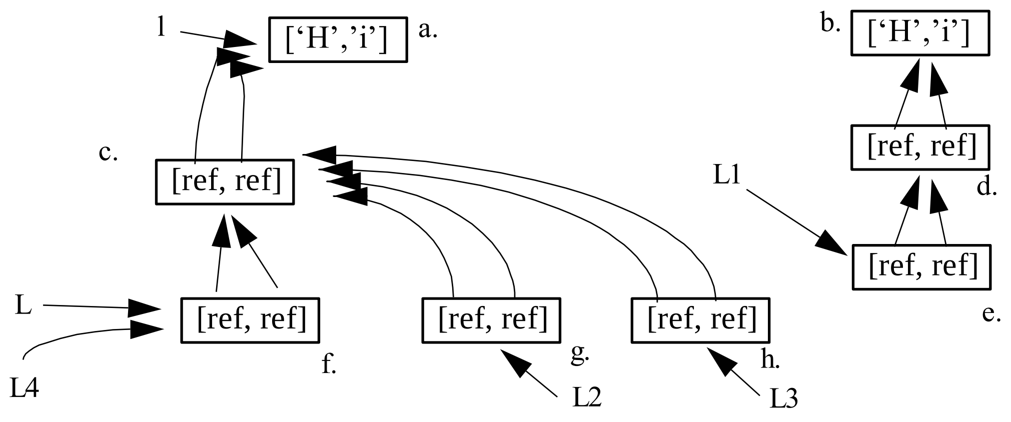
A recursive copy takes place. Note that identical references will point to the same (new) object.

```
>>> L2= copy.copy(L)
>>> L3 = L[:]
```

The objects in the lists are copied to the depth of one.

```
>>> L4 = L
```

Now we have the following situation:



```
>>> L3[0][0][-1:] = []
```

The last object is removed from object `a`.

```
>>> del L3[0];
```

The first object is removed from object `h`.

```
>>> del L3[0][1]
```

The last object is removed from object `c`.
(Note that the original `L3[1]` became `L3[0]` after the previous command)

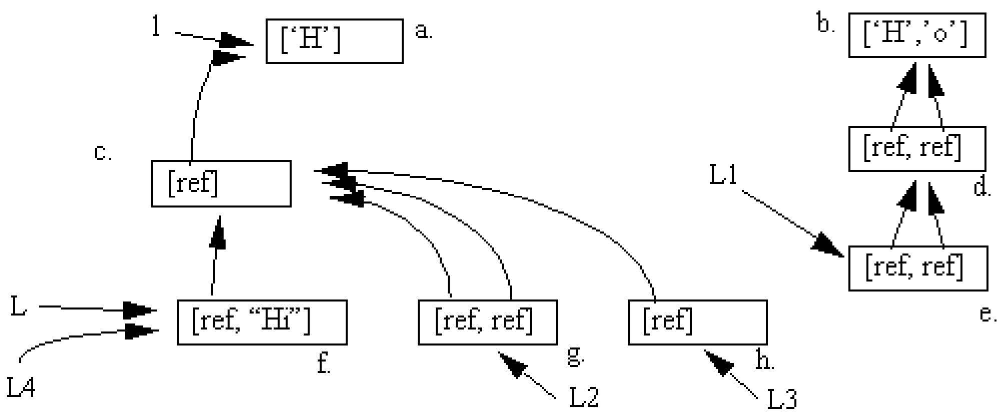
```
>>> L4[1] = "Hi"
```

The second object is replaced in object `f`.

```
>>> L1[1][1][1] = "o"
```

The “`i`” is replaced by an “`o`” in object `b`.

We end up with the following situation



and we get:

```
>>>L1
[[['H', 'o'], ['H', 'o']], [['H', 'o'], ['H', 'o']]]
>>>L2
[[['H']], [['H']]]
>>>L3
[[['H']]]
>>>L4
[[['H']], 'Hi']
```

Exercise 3 Flow control

Exercise 3.1 Manipulate the objects in a list

```
>>> m = l = [2, 4, 5, 6, 7, 7, 8]
```

1. Use a while loop

```
i=0
while i<len(l):
    if not l[i]%2: l[i] += 10
    i+=1
```

2. Use a for loop

```
# Maybe the best solution
for i in range(len(l)):
    if not l[i]%2: l[i] += 10;
```

```
# A one row solution
for i in range(len(l)): l[i] += not l[i]%2 and 10
```

3. Use list comprehension and subscription

```
l[:]=[(x+10,x)[x%2] for x in l]
```

Exercise 3.2 Last visited station

One possible solution:

```
d = {}
for row in file("Large.etab"):
    if row[0] != '"': continue
    d[(row[1:7],row[10:13])] = row[15:24]

for key in (("721111","NRT"),("742246","MIA")):
    print ",".join(key), ":", d[key]
```

Exercise 3.3 Reduce size of external table

The new table should contain 3644 rows of data.

One possible solution:

```
s = set()
for row in file("crew_attributes.etab"):
    if row[0] != '"': continue
    items = row.split(",")
    if items[1][2] != "H": continue
    if items[5][2:-1] in ("LAX","DFW"): s.add(int(items[0][1:-1]))

nf = file("crew_qualification_2.etab","w")
for row in file("crew_qualification.etab"):
    if row[0] != '"' or int(row.split(",")[0][1:-1]) in s:
       nf.write(row)
```

Exercise 4 Functions

Exercise 4.1 Your first function

Step 1

```
def my_add(*a):
    if not a: return None
    b = a[0]
    for val in a[1:]:
        b += val
    return b
```

Step 2 with the reduce function

```
def my_add(*t):
    if not t: return None
    return reduce(lambda a,b:a+b,t)
```

or maybe you prefer a one line solution:

```
my_add=lambda *t:(t and [reduce(lambda a,b: a+b,t)] or [None])[0]
```

Step 3 with recursion

```
def my_add(*t):
    if not t: return None
    if len(t) == 1: return t[0]
    return t[0] + my_add(*t[1:])
```

Exercise 4.2 Manipulate the objects in a list

1. Use the map function

```
def f(x):
    if not x%2 : x += 10
    return x
l[:] = map(f, l)


# Use map function and a lambda expression
l[:]=map(lambda x:(x+10,x)[x%2], l)
```

2. With recursion.

It is hardly recommended to use recursion. Anyway here is a solution:

```
def f(l):
    if not l: return l
    return [l[0] + (10,0)[l[0]%2]] + f(l[1:])
L[:] = f(l)
```

Exercise 4.3 Lambda expressions

Step 1 Filter

```
l[:] = filter(lambda x: x<=10,l)
l[:] = [x for x in l if x<=10]
```

Step 2 Sort

```
l.sort(lambda x,y: cmp(y, x))
```

Another solution:

```
l.sort(reverse=True)
```

Exercise 4.4 Sort

Here we provide two possible solutions:

```
def my_sort(a, b):
    ab = zip(a,b)
    ab.sort(lambda x,y: cmp(x[1], y[1]))
    return [x[0] for x in ab]
def my_sort2(a, b):
    b2= b[:]
    d = dict(zip(b2, a))
    b.sort()
    return [d[x] for x in b2]
```

Exercise 5 Modules

Step 1 Basics

Give the file the name `ex5_1.py`:

```
> python
>>> import ex5_1
```

You should keep the interpreter running.
Change the source code file in another window.

```
>>> import ex5_1
>>> print ex5_1.a
```

You still get the old value.
A module is only imported once by the interpreter.

```
>>> reload(ex5_1)
>>> ex5_1.a
 >>> CTRL-D

> rm $HOME/PYTHON1_course/ex5_1.py
> python
>>> import Ex5_1
```

It works fine because the byte code file (`ex5_1.pyc`) is still there.

Step 2 **`mod1.py`:**

```
"""A test module created to get some basic understanding of
modules in Python"""

if __name__ == "__main__":
    import sys
    print "Do not use %s as script" % sys.argv[0]

_counter = 0

def func():
    global _counter
    _counter += 1
    return(_counter)
```

Step 3 **`mod1test.py`:**

```
"""Just a small test program"""

import mod1

for i in range(5):
    r = mod1.func()

print "mod1.func has been called %s times" % r
```

Step 4

```
> python
>>> import mod1
>>> dir(mod1)
['__builtins__', '__doc__', '__file__', '__name__',
'_counter', 'func']

>>> help(mod1)

Help on module mod1:

NAME

    mod1

FILE

    /users/SID/PYTHON1_course/mod1.py

DESCRIPTION
```

```
    A test module created to get some basic understanding of modules
in Python

FUNCTIONS

    func()

DATA

    __file__ = 'mod1.pyc'

    __name__ = 'mod1'

    _counter = 0
```

Step 5 from and reload

```
> python

>>> import mod1
>>> from mod1 import func as ff
>>> ff(); mod1.func(); mod1.func is ff
1
2
True

>>> reload(mod1)
<module 'mod1' from 'mod1.pyc'>
>>> ff(); mod1.func(); mod1.func is ff
1
2
False
```

After the `reload` there are two function objects. However, they update the same variable so you do not suffer from it.

Step 6 Change the name of the counter in the file `mod1.py`.
(e.g. from "`_counter`" to "`_counter1`")

```
>>> reload(mod1)
<module 'mod1' from 'mod1.py'>
>>> ff(); mod1.func()
3
1
```

After this second reload the functions update different variables.

If you look at the attributes of `mod1` with `dir(mod1)` you see that both the new and the replaced counter are present. Reload does not remove attributes from a module object.

Exercise 6 Classes

Exercise 6.1 A Stack

Step 1 The file `mystack.py:`

```
""" A module containing a very simple stack class """

class Stack(object):
    """ A very simple Stack Class """
    def __init__(self): self._q=[]
    def push(self, item): self._q.append(item)
    def pop(self): return self._q.pop()
```

Restart the interpreter and do `from myStack import Stack` before you enter the test statements specified in the exercises.

Step 2 Add to the class body:

```
    def __str__(self): return "S"+str(self._q)
    def __repr__(self): return "S"+repr(self._q)
```

Step 3 Add to the class body:

```
    def __len__(self):  return len(self._q)
```

Step 4 Add to the class body:

```
def __iadd__(self, item):
    self.push(item)
    return self
```

Step 5 Add an instance counter

```
"""
This is a module containing a very simple stack class.
Version 2 that keep track of the number of living Stacks
"""

class Stack(object):
    _num = 0
    def __init__(self):
        self.__class__._num += 1
        self._q=[]
    def push(self,item): self._q.append(item)
    def pop(self):      return self._q.pop()
    def __len__(self):  return len(self._q)
    def __str__(self):  return "S"+str(self._q)
    def __repr__(self): return "S"+repr(self._q)
    def __iadd__(self, item):
        Stack.push(self, item)
        return self
    def __del__(self): self.__class__._num -= 1

    @classmethod
    def num_instances(cls): return cls._num
```

Exercise 6.2 Last visited station, again

```
class LastVisitedStation(object):

    def __init__(self, filename):

        self.d = {}

        for row in file(filename):

            if row[0] != '"': continue

            self.d[row[1:7], row[10:13]] = row[15:24]

    def get_date(self, crewid, station):

        return self.d[(crewid, station)]

norm_last_visited_stn = LastVisitedStation("Large.etab")

if __name__ == "__main__":
```

```
for key in (("721111","NRT"),("742246","MIA")):
    print ",".join(key), ":", norm_last_visited_stn.get_date(*key)
```

Exercise 6.3 Module Magic

Here is a suggested solution:

```
""" This module keeps track of the number of times it has been
imported.
 Works, mostly, but see
http://stackoverflow.com/questions/1676835/python-how-do-i-
get-a-reference-to-a-module-inside-the-module-itself for
further discussion.
"""

import sys

# Get me a reference to me, myself and I
current_module = sys.modules[__name__]

# Reload counter
if hasattr(current_module, 'reload_counter'):
    reload_counter += 1
    print current_module, "has been reloaded", \
          reload_counter, "time(s)."
else:
    reload_counter = 0
    print "Initial import of", current_module
```

Exercise 7 Exception Handling

Exercise 7.1 Calculator

Create the file `ex_7_1.py`:

```
#!/usr/bin/env python

def my_calc():
    while True:
        expr = raw_input('Expression: ')
        if not expr : break
        try:
            print eval(expr)
        except Exception as err:
            print "Bad input '%s' : %s" % (expr, err)

if __name__ == "__main__":

    my_calc()
```

Make the file executable and run it.

Exercise 7.2 The Stack

```
class EmptyStackError(IndexError):
    "The stack is empty"
```

```
class FullStackError(IndexError):
    "The stack is full"
class Stack(object):
    """ A very simple Stack Class """
    _maxItems = 2

    def __init__(self):
        self._q=[]

    def push(self,item):
        if len(self) >= self._maxItems:
            raise FullStackError("The stack is full")
        self._q.append(item)

    def pop(self):
        if len(self) == 0:
            raise EmptyStackError("No more items")
        return self._q.pop()

    def __str__(self):
        return "S"+str(self._q)

    def __repr__(self):
        return "S"+repr(self._q)

    def __len__(self):
        return len(self._q)

    def __iadd__(self, item):
        self.push(item)
        return self
```

Exercise 8 Iterators & generators

Exercise 8.1 Create generators

Step 1 Use a function having a `yield` statement

```
import sys
def myGen ():
    v = 1
    while v**v < sys.maxint:
        yield v**v
        v+=1
```

Step 2 Use `itertools`. We will use `lambda` to generate fresh copies of `myGen.`

```
import itertools as it
myGen = lambda: it.imap(lambda x: x**x,
                        it.takewhile(lambda x: x**x < sys.maxint,
                                     it.count(1)))
for a in myGen():
    print a,
print
print sum(myGen())
```

Step 3 Without `yield` or `itertools`

```
class myGen(object):
    def __init__(self):
        self.v = 0
    def __iter__(self):
        return self
    def next(self):
        nv = self.v+1
        if nv**nv > sys.maxint:
            raise StopIteration
        self.v = nv
        return nv**nv
```

Exercise 9 Built-in-modules

Exercise 9.1 Weak references

Step 1 Read about the `weakref` module.

Step 2 Test

```
>>> s1 = Stack(); s2 = Stack()
>>> print Stack.num
>>> s1.push(s2); s2.push(s1); print Stack.num
2
>>> del s1, s2; print Stack.num
2
```

Step 3 Use `weakref`

```
>>> import weakref
>>> s1=Stack(); s2=Stack()
>>> s1.push(s2); s2.push(weakref.ref(s1))
>>> del s1,s2; print Stack.num
0
```

Now it works fine!

Exercise 9.2 Sub process

-

Exercise 9.3 Performance

```
> python -mtimeit -s "l=[]" "for dum in range(10000): l.insert(0, 0)"

> python -mtimeit -s "l=[]" "for dum in range(10000): l[0:0]=[0]"

> python -mtimeit -s "l=[]" "for dum in xrange(10000): l[0:0]=[0]"

> python -mtimeit -s "l=[]" "for dum in xrange(10000): l.append(0)"

> python -mtimeit "[0 for dum in xrange(10000)]"

> python -mtimeit -s "import itertools as it" \
?                    "[0 for dum in it.repeat(None, 10000)]"

> python -mtimeit -s "[0]*10000"
```

The first three are about 10000 times slower than the last one.

Why?

Lists are implemented as arrays. Adding a new object in the end is much faster than inserting. When you insert you have to shift everything in the list.

`[0]*10000` allocates all the required memory and fills it with `0` without any iteration at all and it is very fast.

The rest is not that important:

- if you use the insert method or slicing has no importance
- which iterator you use is not critical
- list comprehension is faster than the `for` loop but not that significant.

Exercise 9.4 Doctests

There is really no right or wrong way of doing this – either your tests pass or they don't; one should always aim for completion, though, so the more extensive the test, the more potential errors you will cover.

Below is a simple doc test that will cover the examples in exercise 6.1; it is overly verbose on purpose:

```
======================
Tests for exercise 6.1
======================

We want to create a class stack that offers the methods pop()
and push(item).

>>> from ex61 import *
>>> s = Stack()
>>> s
S[]

Let's push and pop some items on the stack:

>>> s.push(1)
>>> s.push(2)
>>> s
S[2, 1]
>>> s.pop()
2
>>> s
S[1]
>>> s.pop()
1
>>> s
S[]

Let's push some stuff back into the stack and get the length
of it:

>>> s.push(1)
>>> s.push(2)
>>> len(s)
2
>>> s
S[2,1]

Another way to put stuff into the stack:

>>> s += 3
>>> s
S[3, 2, 1]

Let's see how many Stack instances we have.

>>> s.num_instances()
```

```
1
>>> s2 = Stack()
>>> Stack.num_instances()
2
>>> s.num_instances()
2
>>> del s
>>> s2.num_instances()
1
>>> del s2
>>> Stack.num_instances()
0

# Done with test
```

Exercise 9.5 Unittests

The same as above, but as unit tests, saved as "`test_ex61.py`" in the same directory as `ex61.py`:

```
import unittest
import ex61

class StackTest(unittest.TestCase):
    def testStackCreation(self):
        s = ex61.Stack()
        self.assertEqual(s.__repr__(), 'S[]')

    def testStackPush(self):
        s = ex61.Stack()
        s.push(1)
        s.push(2)
        self.assertEqual(s.__repr__(), 'S[2, 1]')

    def testStackPop(self):
        s = ex61.Stack()
        s.push(1)
        s.push(2)
        x2 = s.pop()
        x1 = s.pop()
        self.assertEqual(x2, 2)
        self.assertEqual(x1, 1)
        self.assertEqual(s.__repr__(), 'S[]')

    def testStackLength(self):
        s = ex61.Stack()
        s.push(1)
        s.push(2)
        self.assertEqual(len(s), 2)

    def testStackIAdd(self):
        s = ex61.Stack()
        s += 1
        s += 2
        self.assertEqual(s.__repr__(), 'S[2, 1]')
        x2 = s.pop()
        x1 = s.pop()
        self.assertEqual(x2, 2)
        self.assertEqual(x1, 1)
        self.assertEqual(s.__repr__(), 'S[]')

    def testStackNumInstances(self):
        s = ex61.Stack()
        s += 1
        s += 2
        s2 = ex61.Stack()
        s2 += 1
        self.assertEqual(s.num_instances(), 2)
        self.assertEqual(s2.num_instances(), 2)
        self.assertEqual(ex61.Stack.num_instances(), 2)
        del s
        self.assertEqual(s2.num_instances(), 1)
        self.assertEqual(ex61.Stack.num_instances(), 1)
        del s2
        self.assertEqual(ex61.Stack.num_instances(), 0)

if __name__ == '__main__':
    unittest.main()
```

Exercise 9.6 Find the largest files

Create an executable file:

```
#!/usr/bin/env python
#
# Exercise Build-in-modules 4
#
"""
This script prints the largest files in a directory tree.

Supported flags:
        -h,-H      : this help
        -n NUM     : number of files to print. Default 10
        -d DIR     : top directory. Default current
        -o OWNER   : only files owned by OWNER
"""

import os, sys, pwd, getopt, subprocess

def getLargeFiles1(top_dir=".", show_num=10, only_user=None):
    if only_user: only_uid = pwd.getpwnam(only_user).pw_uid
    large_files = []
    for dpath,dirnames,filenames in os.walk(top_dir):
        for fn in filenames:
                path = os.path.join(dpath, fn)
                if os.path.islink(path): continue
                if only_user and os.stat(path).st_uid != only_uid: continue
                large_files.append((os.stat(path).st_size,path))

    large_files.sort(reverse=True)
    del large_files[show_num:]
    return "\n".join(["%10s %s" % t for t in large_files])

def getLargeFiles2(top_dir=".", show_num=10, only_user=None):
    command1 = """find %s %s -type f -printf "%%10s %%p\n" | sort -n -r | head -
%s"""
    command = command1 % (top_dir,
                          only_user and "-user "+only_user or "",
                          show_num)
    return subprocess.Popen(command,shell=True,
                           stderr=subprocess.PIPE,
                           stdout=subprocess.PIPE).communicate()[0][:-1]

if __name__ == "__main__":
    def do():
        try:
            opts, res = getopt.getopt(sys.argv[1:], "n:d:o:hH")
        except getopt.GetoptError, err:
            print err
            print "For info about options use -h"
            return
        if res:
            print "No arguments accepted."
            print "Use -h as flag for information about options"
            return

        args = {}
        for opt in opts:
            if opt[0] == "-n": args["show_num"]  = int(opt[1])
            if opt[0] == "-d": args["top_dir"]   = opt[1]
            if opt[0] == "-o": args["only_user"] = opt[1]
            if opt[0].upper() == "-H":
                print __doc__
                return
        print getLargeFiles1(**args)  # Change function to test the other one.
    do()
```

Exercise 9.7 Profiler

Do some changes in the script:

```
#!/usr/bin/env python
#
# Exercise Build-in-modules 5
#
"""
This script prints the largest files in a directory tree.

Supported flags:
       -h,-H     : this help
       -n NUM    : number of files to print, default 10
       -d DIR    : top directory, default current
       -o OWNER  : only files owned by OWNER
       -p        : Use profiler
       -2        : Use the second implementation
"""

import os, sys, pwd, getopt, subprocess, tempfile
import profile, pstats

def getLargeFiles1(top_dir=".", show_num=10, only_user=None):
    if only_user: only_uid = pwd.getpwnam(only_user).pw_uid
    large_files = []
    for dpath,dirnames,filenames in os.walk(top_dir):
        for fn in filenames:
                path = os.path.join(dpath, fn)
                if os.path.islink(path): continue
                if only_user and os.stat(path).st_uid != only_uid: continue
                large_files.append((os.stat(path).st_size, path))

    large_files.sort(reverse=True)
    del large_files[show_num:]
    return "\n".join(["%10s %s" % t for t in large_files])

def getLargeFiles2(top_dir=".", show_num=10, only_user=None):
    command1 = """find %s %s -type f -printf "%%10s %%p\n" | sort -n -r | head -
%s"""
    command = command1 % (top_dir,
                          only_user and "-user "+only_user or "",
                          show_num)
    return subprocess.Popen(command, shell=True,
                           stderr=subprocess.PIPE,
                           stdout=subprocess.PIPE).communicate()[0][:-1]

if __name__ == "__main__":
    import __main__ as m
    def do():
        try:
            opts, res = getopt.getopt(sys.argv[1:], "n:d:o:phH2")
        except getopt.GetoptError, err:
            print err
            print "For info about options use -h"
            return
        if res:
            print "No arguments accepted."
            print "Use -h as flag for information about options"
            return

        prof = False
        func_name = "getLargeFiles1"

        args = {}
        for opt in opts:
            if opt[0] == "-n": args["show_num"]  = int(opt[1])
            if opt[0] == "-d": args["top_dir"]   = opt[1]
            if opt[0] == "-o": args["only_user"] = opt[1]
            if opt[0] == "-p": prof = True
            if opt[0] == "-2": func_name="getLargeFiles2"
            if opt[0].upper() == "-H":
                print __doc__
                return

        if prof:
            f,fn = tempfile.mkstemp()
            profile.run("s=%s(**%s)"%(func_name, args), fn)
            print s
            stats = pstats.Stats(fn)
            os.unlink(fn)
            stats.sort_stats('time', 'calls')
            stats.print_stats(20)
        else:
            print getattr(m, func_name)(**args)
    do()
```

Note The CPU time spent in sub processes is not considered.

Quick reference

Keywords:

`and`	`del`	`for`	`is`	`raise`
`assert`	`elif`	`from`	`Lambda`	`return`
`Break`	`else`	`global`	`not`	`try`
`Class`	`except`	`if`	`or`	`while`
`continue`	`exec`	`import`	`pass`	`yield`
`Def`	`finally`	`in`	`print`	

Numbers:

```
1, 42, 3.1415, 1.0J
```

Texts:

```
‘Hello’,
"World",
"""Jeppesen
Systems AB"""
```

List:

```
[1, "Jeppesen", (1,2,3)]
```

Tuple:

```
(101, 102, 103)
```

Dictionary:

```
{‘x’:"Jeppesen", ‘y’:"Systems", ‘z’:"AB"}
```

Files:

```
fh = open(filename[,mode=’r’, bufsize=-1])
fh.close()
```

if:

```
if prod > 80:
    print "Good Productivity"
elif x < 40:
    print "Not so good..."
else: print "Productivity %s" % x
```

while:

```
x = 11
while x > 0:
    x -= 1
    if x % 2: continue    # only print even numbers
    print 10 * x
    if not keep_looping(x): break
```

for:

```
for l in "Optimization Matters":
   print l.upper(),
```

```
OPTIMIZATION MATTERS
```

range / xrange:

```
for i in xrange(10, 0, -1):
    print i*10,
```

```
100 90 80 70 60 50 40 30 20 10
```

functions:

```
def my_sum_func(start=0, stop=10):
    sum = 0
    for i in xrange(start, stop+1):
        sum += i
    return sum
```

```
print my_sum_func(10, 11)
```

```
21
```

lambda:

```
lambda parameters: expression
```

map:

```
map(func, seq, *seqs)
```

```
L = [1,2,3,4,5,6,7]
L[:] = map(lambda x:(x+10, x)[x%2], L)
[1, 12, 3, 14, 5, 16, 7]
```

filter:

```
filter(func, seq)
```

```
filter(lambda char: char.isupper(), "Jeppesen Systems AB")
‘JSAB’
```

reduce:

```
reduce(func, seq [,init])
```

```
import operator
def my_sum_func2(start=0, stop=2):
    return reduce(operator.add, range(start, stop+1) )
```

```
my_sum_func2()
3
```

List comprehension:

```
[expression for target in iterable if expr]
```

```
quad = [x**2 for x in range(3)]
```

```
[0,1,4]
```

Modules:

```
import modname [as varname]
(from modname import attrname [as varname])
```

```
reload(modname)
```

```
(from MyModule import *)  AVOID!
```

Classes:

```
class name [(classes)]:
    statements(s)
```

```
class Crew(object):
    def __init__(self, start): self._salary = start
    def increase(self, amount): self._salary += amount
```

```
class Pilot(Crew):
    def __init__(self, start, qual):
        self._salary = start
        self._qualificaton = qual
```

```
Iceman = Pilot(100000, "Star")
```

Exceptions:

```
try:
    statements(s)
except [expression [,target]]:
    statements(s)
[else:
    statements(s)]
```

```
try:
    statements(s)
finally:
    statements(s)
```

```
def safe_divide(x, y):
    try:
        return x/y
    except ZeroDivisionError:
        print "Div by Zero Error"
        return None
```

```
f = open(myFile, "w")
try:
    try_to_read_file(f)
finally:
    f.close()
```

```
class FullStackError(IndexError):
    "The stack is full"
```

```
if len(self) >= self._maxItems:
    raise FullStackError("Stack is full")
```

Generator:

```
def  myrepeat(num):
    while num:
          num -= 1
          yield True
```

Generator expression:

```
sum(x*x for x in xrange(5))
```