Contents

1 Introduction 3

2 Bash 3
  2.1 Variables 3
  2.2 Loops 4
    2.2.1 For loop 4
  2.3 Pipes 4
  2.4 Functions 5

3 Basic Python 5
  3.1 Variables 5
  3.2 Lists and tuples 6
  3.3 Strings 6
  3.4 Conditionals 7
  3.5 Loops 7
  3.6 Functions 7
  3.7 Read from file 8
  3.8 Using modules 8
  3.9 Classes 8

4 Stuff that is nice to know 9
  4.1 Copy, rename and remove files 9
  4.2 Traverse directory tree 9
1 Introduction

I took INF3331 - Problem solving with high-level languages in the fall 2013. These notes consists of a summary of my weekly assignments, lectures slides and resources online.

2 Bash

Use plain Bourne shell (/bin/sh) when special features of Bash (/bin/bash) are not needed.

2.1 Variables

Variables in Bash are untyped.

```
1 x = 3
2 y = 2
3 z = $x+$y
4 echo $z       # output 3+2

6 z = $(x+y)
7 ((v=x+y))
8 let w = x+y
9 echo $z $v $w  # output 5 5 5
```

Handle variables as they where strings, unless they are explicit declared as an Integer:

```
1 declare -i x = 3
```

Example of handling as string:

```
1 if ["$?" !="0"]; then    # safe
2 if [$? != 0]; then    # may be unsafe
```

Command line arguments are accessed like this:

```
1 $1 $2 $3 $4       # and so on
```

And they are initialised as this:
You may also store Unix-commands in variables:

```bash
s = 'echo "s($r)" | bc -1'
```

or

```bash
s = $(echo 's($r)' | bc -1')
```

### 2.2 Loops

#### 2.2.1 For loop

Following script is removing all `.tmp`-files at its location.

```bash
files = '/bin/ls *.tmp'
# use /bin/ls in case ls is aliased
for file in $files
do
echo removing $file
rm -f $file
done
```

C-style for loop:

```bash
declare -i i
for ((i=0; i<$n; i++)); do
echo $c
done
```

### 2.3 Pipes

Output from one command can be sent as input to another through a pipe.

For sending files with size to reverse numerical sort (`-rn`) and get a list of files sorted after size:

```bash
/bin/ls -s | sort -r
```


2.4 Functions

Example of a function with argument:

```c
function quit {
exit
}
function e {
    echo $1
}
e Hello  # calling e with argument Hello
e World  # calling e with argument World
quit     # calling quit
```

3 Basic Python

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

print "Hello, World!"

../code/hw.py
```

To run Python, simply type `~$ python yourprogram.py <argument1> <argument2> ...`

3.1 Variables

Variables in Python hold references to objects of any type.

```c
a = 3
a = 3.0
a = '3'
a = ['1', 2]

# test for var type
if isinstance(a, int):  # int?
    if isinstance(a, (list, tuple)):  # list or tuple?
```

Common types in Python:

- Numbers: int, float, complex
- Sequences: str (string), list, tuple, NumPy array
• Mappings: dict (dictionary/hash)
• User-defined type in terms of a class

3.2 Lists and tuples

```python
mylist = ['a', 2.5, 42, 'b']
mytuple = ('a', 2.5, 42, 'b')
mylist[1] = -10
mylist.append('c')
mytuple[1] = -10  # illegal, cannot change a tuple
```

List functionality

```python
a = []  # init empty list
a = [1, 'b']  # init a list
a.append(e)  # add e to end of list
a + [1,3]  # add two lists
a[3]  # index a list element
a[-1]  # get last list element
a[1:3]  # copy data to sublist, here: index 1,2
del a[3]  # delete element at index 3
a.remove(1)  # remove element with value 1
a.index('b')  # find index with value 'b'
'b' in a  # test if value 'b' exist in list
a.count(v)  # count number of elements with value v
len(a)  # number of elements in a
min(a)  # smallest element in a
max(a)  # largest element in a
sum(a)  # add all elements in a
a.sort()  # sort list a (changes a)
as = sorted(a)  # sort list a (returns new list)
a.reverse()  # reverse list a (changes a)
b[3][0][2]  # nested list indexing
```

3.3 Strings

```python
s1 = "some string with number %g" % r
s2 = 'another string'
text = """
here you may write
```
# String operations

```python
s = 'some#string somestring'
s[8:10] # extract substring
s.find('i') # index where 'i' is found
s.split('#') # split into substrings
'some' in s # test if substring is in s
s.replace('i', 's') # lower case only
s.upper() # upper case only
s.strip() # remove blanks
', ' .join(s1)
```

../code/str.py

## 3.4 Conditionals

```python
if condition:
    # block of statements
elif condition:
    # block of statements
else:
    # block of statements
```

../code/con.py

## 3.5 Loops

```python
while condition:
    # block of statements
for element in somelist:
    # block of statements
```

../code/loop.py

## 3.6 Functions

```python
def fib(n): # write Fibonacci series up to n
    """Print a Fibonacci series up to n"
```

../code/func.py
```python
a, b = 0, 1
while b < n:
    print b,
    a, b = b, a+b

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

### 3.7 Read from file

```python
# With for-loop
file = open('newfile.txt', 'r')
for line in file:
    print line

# With statement
with open("newtext.txt") as file:
    data = file.read()
    # do something with data

# Read into list
with open(hello.txt) as f:
    data = f.readlines()
```

### 3.8 Using modules

```python
import sys

# Will store first command line argument in x
x = float(sys.argv[1])

# Same shit:
from sys import argv
x = float(argv[1])
```

### 3.9 Classes

All functions are virtual. No private or protected variables. self is a reference to this object. Data members are prefixed by self.
```python
class MyClass:
    def __init__(self, i, j):  # ctor
        self.i = i; self.j = j

    def write(self):
        print 'MyClass: i=', self.i, ', j=', self.j

../code/simpleclass.py
```

4 Stuff that is nice to know

4.1 Copy, rename and remove files

```python
# Copy a file
import shutil
shutil.copy(myfile, tmpfile)

# Rename a file
import os
os.rename(myfile, 'tmp.1')

# Remove a file
os.remove('mydata')
../code/crr.py
```

4.2 Traverse directory tree

```python
import os

for dirname, dirnames, filenames in os.walk('.'):  # print path to all subdirectories first.
    for subdirname in dirnames:
        print os.path.join(dirname, subdirname)

    # print path to all filenames.
    for filename in filenames:
        print os.path.join(dirname, filename)
../code/trav.py
```
5 Regular expression

5.1 Special chars

.  any single character except newline
\^ the beginning of the line or string
\$ the end of line or string
*  zero or more of the last character
+  one or more of the last character
?  zero or one of the last character
\[A-Z\] match all upper case letters
\[abc\] match either a or b or c
\[^b\] does not match b
\[^a-z\] does not match lower case
.\.* any sequence of characters (except newline)
\[.*\] the characters . and *
\^no the string 'no' at the beginning of a line
\[^no\] neither n or o
A-Z the 3-character string 'A-Z'
(eg |le) gs match eggs or legs

5.2 Common expressions

\n a newline
\t a tab
\w any alphanumeric (word) character, the same as \[a-zA-Z0-9\]
\W any non-word character, the same as \[^a-zA-Z0-9\]
\d any digit, the same as \[0-9\]
\D any non-digit, the same as \[^0-9\]
\s any white space character: space, tab, newline etc.
\S any non-white space character
\b a word boundary, outside [] only
\B no word boundary

5.3 Quoting special characters
\. a dot
\| a vertical bar
\[ a open square bracket
\) a closing parenthesis
\^ a hat
\\ a backslash
    : And so on..

5.4 Handy functions

```python
import re

# compile regex pattern to regex object
# to be used for match() and search()
re.compile(pattern)

# scans string for a location where regex
# produces a match. Returns MatchObject.
re.search(pattern, string)

# if zero or more chars at beginning of string
# the regex returns a corresponding MatchObject.
re.match(pattern, string)

# split string by the occurrences of pattern
re.split(pattern, string)
```
17 # returns all non-overlapping matches of
18 # pattern in string, as a list of strings.
19 re.findall(pattern, string)
20 ../code/regexfunc.py

5.5 Examples of regex

5.5.1 Scientific or decimal or integer notation

1 # One-liner:
2 pattern = r’−?(\d\.?\d*[Ee][+−]?)\d+|(%d+|\d*\.\d+)\d+’
3
4 # Modularized:
5 real_in = r’\d+’
6 real_dn = r’(%d+|\d*\.\d+)’
7 real_sn = r’(%d\.?\d*[Ee][+−]?)\d+’
8 real = ’−?(’ + real_sn + ’|’ + real_dn + ’|’ + real_in + ’)
9 ../code/regexnum.py

6 Numerical Python

NumPy enables efficient numerical computing in Python. This is a package of
modules, which offers efficient arrays with associated array operations coded
in C or Fortran.

1 from numpy import *

6.1 Operations on NumPy arrays

1 array() # creates a new array
2 zeros(x) # fills the new array with x zeros
3 shape(v,w) # turns it into a v times w matrix
4 reshape(v,w) # returns an array with the same
5 # data, but a new shape
6 fill(x) # fill array with scalar value x
7 copy # returns a copy of the array
8
9 # Basic array indexing
11 a[-1] = a[0]  # set last elem equal to first elem
12 a[:] = 0  # set all elements to 0
13 a[i,j] = 10  # assignment to element (i,j)
14 print a[:,k]  # print column with index k
15 print a[1,:]  # print second row
16 print a[0,1]  # print element (0,1)

6.2 Example of Python and NumPy

6.2.1 Vectorization of a function

1 #!/usr/bin/env python
2 from numpy import *
3
4 def initial_cond(x):
5     if type(x) == type(array([])):
6         result = zeros(x.shape)
7         result.fill(3.0)
8         return result
9
10 if __name__ == "__main__":
11     print initial_cond(array(zeros(6)).reshape(3,2))

7 Web applications in Python

7.1 CGI

Example with Python, HTML and CGI. Computes sine of a number from user input.

1 <html>
2     <form action="hwhtml.py.cgi" method="post">
3         Hello, World! The sine of
4         <input type="text" name="r" size="10" value="1.2">
5         <input type="submit" value="equals" name="equalsbutton">
6     </form>
7 </html>

../code/hw.html
importcgi,math
form=cg.FieldStorage()
r=form.getvalue('r')
if r is not None:
s=str(math.sin(float(r))
else:
s='';r=''

Do not bother to start with CGI. Rather check out Django.

References


[2]Python documentation