Input

raw_input(parameter) - Escreve o parâmetro como um prompt e recebe uma string.

Try and Except

Usamos isso numa parte de código que pode dar problem.

```
astr = 'Hello Bob'
istr = int(astr)

Solução:
Try:
    istr = int(astr)
Except:
    istr = -1
```

Se utilizamos um bloco de código dentre de um try, ele roda até encontrar um que falhe

```
try:
    print 'Hello'
    istr = int(astr)
    print 'There'
except:
    istr = -1
```

Para abandonar um programa no meio eu preciso utilizar o comando:

```
quit()
```

Função

```
def nome_da_função(argumentos):
    algo
    return
```

break - quebra um loop while e sai dele

```
while True:
    line = raw_input('>')
    if line == 'done':
        break
    print line
print 'Done'
```

continue - quebra a execução, mas volta para o início do loop

```
while True:
    line = raw_input('>')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print line
print 'Done'
```

while - indefinite loop
for - definite loop

```
for i in array:
    print i
```

String

slicing
s[0:4], retorna caracteres de 0 a 4 exclusive
s[:2], retorna do começo da string até 2 exclusive
s[8:], retorna da posição 8 até o final da string
s[:], retorna toda a string

`if 'a' in variable`, verifica se há 'a' na string

dir(objeto) lista os métodos associados a um objeto.

s.find('other'), procura uma substring. Retorna o índice.
s.replace('old','new'), retorna uma string substituindo old por new
s.lstrip(), elimina espaços brancos à esquerda
s.rstrip(), elimina espaços brancos à direita
s.strip(), elimina todos os espaços brancos
startswith(), verifica se começa com o valor entre parêntesis
split('delimitador') -> quebra uma string pelo delimitador e devolve em uma lista

Se o delimitador não for especificado, ele trata vários espaços como um só.

Files

open(filename,mode)
\n - new line, conta como um caracter apenas

Como ler um arquivo.
Modo 1, linha a linha

```
fhand = open('file.txt')
for line in fhand:
    print line
```

Modo 2, o arquivo inteiro:

```
fhand = open('file.txt')
inp = fhand.read()
print inp
```

Tirando o fim de linha e imprimindo em tela só o que interessa.

Modo 1:

```
fhand = open('file.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:')
        print line
```

Modo 2:

```
fhand = open('file.txt')
for line in fhand:
    line = line.rstrip()
    # Skip 'uninteresting' lines
    if not line.startswith('From:')
        continue
    # Process our 'interesting' line
    print line
```

Modo 3:

```
fhand = open('file.txt')
for line in fhand:
    line = line.rstrip()
    if not '@uct.ac.za' in line
        continue
    print line
```

Tupla

Tupla é uma lista imutável

x = ('a','b','c')

Não dá para fazer sort, não dá para fazer append, não dá para inverter (reverse)
Dá para contar e utilizar index

Como elas não são mutáveis elas usam uma estrutura mais enxuta o que as torna mais rápidas.

Variáveis temporárias, por exemplo.

(x, y) = (4, 5) # duplo assignment

Pode ser feito sem parêntesis

x, y = (4, 5)

Quando listamos um dicionário com o loop que utilizamos antes, ele mostra uma lista de tuplas.

```
for a, b in names.items():
    print a, b
```

Tuplas são comparáveis, >, <, =

Comparam item a item em ordem

Como as listas são comparáveis, uma lista com tuplas de um dicionário pode ser ordenada com sort (sendo key o índice).

Também podemos utilizar a função sorted que recebe como parâmetro uma lista e devolve uma lista organizada.

```
for k, v in sorted(names.items()):
    print k, v
```

Imprime ordenando por k.

Ordenar por valor:

If we could construct a list of tuples of the form (value, key) we could sort by value
We do this with a for loop that creates a list of tuples

```
c = {'a':10, 'b':1, 'c':22}
tmp = list()

for k, v in c.items() :# a tupla foi criada com o valor no primeiro item e a chave depois
    tmp.append( (v, k) )

tmp.sort(reverse=True)
```

Como criar uma lista ordenada de itens de acordo com o maior valor a partir de um dicionário

```
fhand = open('romeo.txt')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0 ) + 1
lst = list()
for key, val in counts.items():
    lst.append( (val, key) )
lst.sort(reverse=True)
for val, key in lst[:10] :
    print key, val
```

Maneira enxuta de criar uma lista ordenada

```
c = {'a':10, 'b':1, 'c':22}
print sorted( [ (v,k) for k,v in c.items() ] )
```

Os símbolos [] indicam para o Python a construção dinâmica de uma lista.

Regular Expression

```
^        Matches the beginning of a line
$        Matches the end of the line
.        Matches any character
\s       Matches whitespace
\S       Matches any non-whitespace character
*        Repeats a character zero or more times
*?         Repeats a character zero or more times
(non-greedy)
+        Repeats a character one or more times
+?         Repeats a character one or more times
(non-greedy)
[aeiou]   Matches a single character in the listed
set
[^XYZ]     Matches a single character not in the
listed set
[a-z0-9] The set of characters can include a range
(          Indicates where string extraction is to
start
)          Indicates where string extraction is to
end
```

re.search() devolve a primeira ocorrência
re.findall() devolve todas as ocorrências numa lista

```
x = 'From: Using the : character'
y = re.findall('^F.+:', x)
print y
['From: Using the :']
```

Ele pega a maior ocorrência. Greedy.

Para parar na primeira ocorrência, usamos um ?.

```
y = re.findall('^F.+:?', x)
```

Request and Response

socket - é uma conexão. Ele é o equivalente a um file handle.

Cada porta é o endereço de um serviço.

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect((‘www.py4inf.com’, 80))
```

Opens a connection, sends a get request, retrieves the data, and the connection gets closed.

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect((‘www.py4inf.com’, 80))

mysock.send('GET           http://www.py4inf.com/code/romeo.txt
HTTP/1.0\n\n')
while True:
    data = mysock.recv(512)
    if ( len(data) < 1 ) :
        break
    print data
mysock.close()
```

socket é uma biblioteca de baixo nível. urllib faz o mesmo em alto nível.

```
import urllib
fhand = urllib.urlopen('http://www.py4inf.com/code/romeo.txt')

for line in fhand:
    print line.strip()
```

Beautiful soup

```
import urllib
from BeautifulSoup import *

url = raw_input('Enter - ')
html = urllib.urlopen(url).read()
soup = BeautifulSoup(html)

# Retrieve a list of the anchor tags
# Each tag is like a dictionary of HTML attributes

tags = soup('a')
for tag in tags:
   print tag.get('href', None)
```

```
import xml.etree.ElementTree as ET

data = '''
<person>
  <name>Chuck</name>
  <phone type="intl">
     +1 734 303 4456
   </phone>
   <email hide="yes"/>
</person>'''

tree = ET.fromstring(data)
print 'Name:',tree.find('name').text
print 'Attr:',tree.find('email').get('hide')
```

JavaScript Object Notation

```
import json
data = '''{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
   },
   "email" : {
     "hide" : "yes"
} }'''

info = json.loads(data)
print 'Name:',info["name"]
print 'Hide:',info["email"]["hide"]
```

Pode começar com [] e assim representa uma lista. O exemplo anterior é apenas um dicionário.

```
import json
input = '''[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
},
{ "id" : "009",
"x" : "7",
    "name" : "Chuck"
  }
]'''
info = json.loads(input)
print 'User count:', len(info)
```

```
for item in info:
    print 'Name', item['name']
    print 'Id', item['id']
    print 'Attribute', item['x']
```