Προγραμματισμός σε C++ & Python & Εφαρμογές στη Ναυπηγική & Ναυτική Μηχανολογία

ΣΝΜΜ 2019

Μάθημα 2A

Γεώργιος Παπαλάμπρου
Επίκουρος Καθηγητής ΕΜΠ
george.papalambrou@lme.ntua.gr

Εργαστήριο Ναυτικής Μηχανολογίας (Κτίριο Λ)
Σχολή Ναυπηγών Μηχανολόγων Μηχανικών
Εθνικό Μετσοβιο Πολυτεχνείο

March 6, 2019

Περιεχόμενα

Περιεχόμενο Μαθήματος

- Εβδομάδα 1. A. Εισαγωγή. Η γλώσσα. Το περιβάλλον Linux. Command line. Python interpreter. Ιστοσελίδα μαθήματος. Βιβλιογραφία. Editors: Sublime, Spyder
  B. Εισαγωγή στην γλώσσα Python. Hello World.
- Εβδομάδα 2. A. Data types. Loops. Conditionals. File I/O
  B. Παραδείγματα.
- Εβδομάδα 3. A. Functions. Modules
  B. OOP. Classes
- Εβδομάδα 4. A. Βιβλιοθήκες NymPy, SciPy. Errors-Exceptions
  B. Παραδείγματα: Γραμμική άλγεβρα, Γραφικά
- Εβδομάδα 5. Εφαρμογή: Neural Networks. Machine Learning
- Εβδομάδα 6. Εφαρμογή: Hardware. Πλατφόρμες. Πρωτόκολλα. Βασικό I/O

Εισαγωγή

Η παράδοση προέρχεται από:

- Core Python Programming (2nd Ed.), Wesley Chun

Προυπόθεση (έχετε δει):

- Chapter 2. Getting started

Για σήμερα:

- Chapter 3. Python Basics
- Chapter 5. Numbers
- Chapter 8: Conditionals and Loops
- Chapter 9: Files and Input/Output

Είσοδος δεδομένων (Program Input)

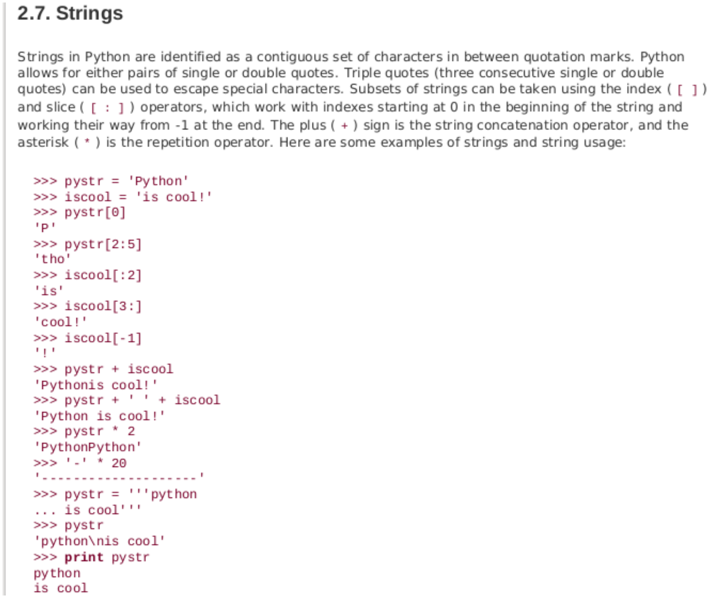
- Στην περίπτωση που ζητείται από τον χρήστη να δώσει είσοδο από την μονάδα εισόδου (standard input) δίνουμε `raw_input()`. Μόνο για Python 2.

```
 >>> user = raw_input('Enter login name:  ')
Enter login name:  root
>>> print 'Your login is:', user
Your login is:  root
```

- Στην Python 3 δίνουμε `input()` *και* print()

```
 >>> user = input('Enter login name:  ')
Enter login name:  papalambrou
>>> print('Your login is:', user)
Your login is:  papalambrou
```

Χαρακτήρες (Strings)



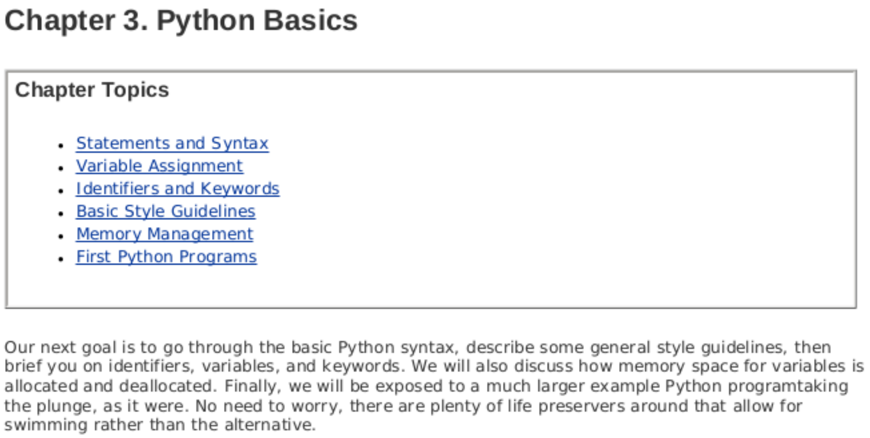
2.7. Strings

Strings in Python are identified as a contiguous set of characters in between quotation marks. Python allows for either pairs of single or double quotes. Triple quotes (three consecutive single or double quotes) can be used to escape special characters. Subsets of strings can be taken using the index ([ ]) and slice ([:]) operators, which work with indexes starting at 0 in the beginning of the string and working their way from -1 at the end. The plus (+) sign is the string concatenation operator, and the asterisk (*) is the repetition operator. Here are some examples of strings and string usage:

```
>>> pystr = 'Python'
>>> iscool = 'is cool!'
>>> pystr[0]
'P'
>>> pystr[2:5]
'tho'
>>> iscool[:2]
'is'
>>> iscool[3:]
'cool!'
>>> iscool[-1]
'!'
>>> pystr + iscool
'Pythonis cool!'
>>> pystr + ' ' + iscool
'Python is cool!'
>>> pystr * 2
'PythonPython'
>>> '-' * 20
'--------------------'
>>> pystr = '''python
... is cool'''
>>> pystr
'python\nis cool'
>>> print pystr
python
is cool
```

Python: Τα Βασικά

Επισκόπηση



Chapter 3. Python Basics

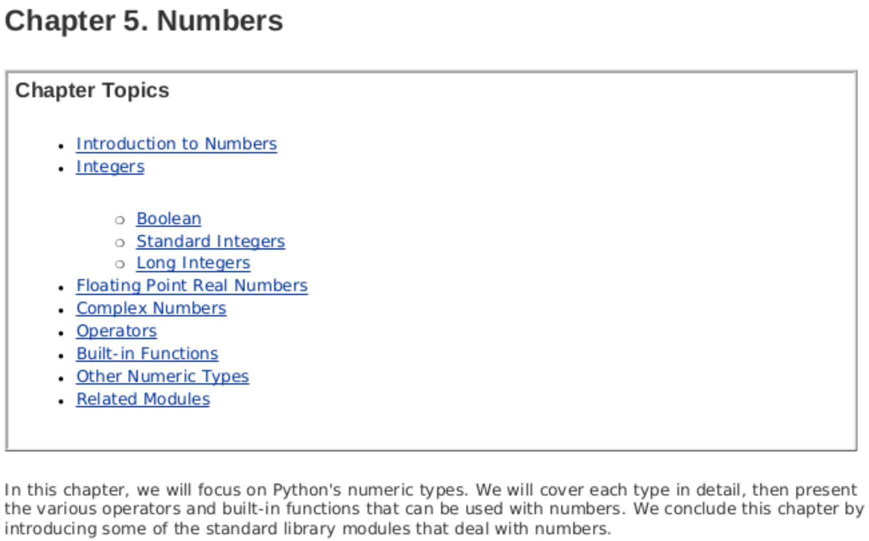
Chapter Topics

- Statements and Syntax
- Variable Assignment
- Identifiers and Keywords
- Basic Style Guidelines
- Memory Management
- First Python Programs

Our next goal is to go through the basic Python syntax, describe some general style guidelines, then brief you on identifiers, variables, and keywords. We will also discuss how memory space for variables is allocated and deallocated. Finally, we will be exposed to a much larger example Python programtaking the plunge, as it were. No need to worry, there are plenty of life preservers around that allow for swimming rather than the alternative.

Python: Οι Αριθμοί

Επισκόπηση

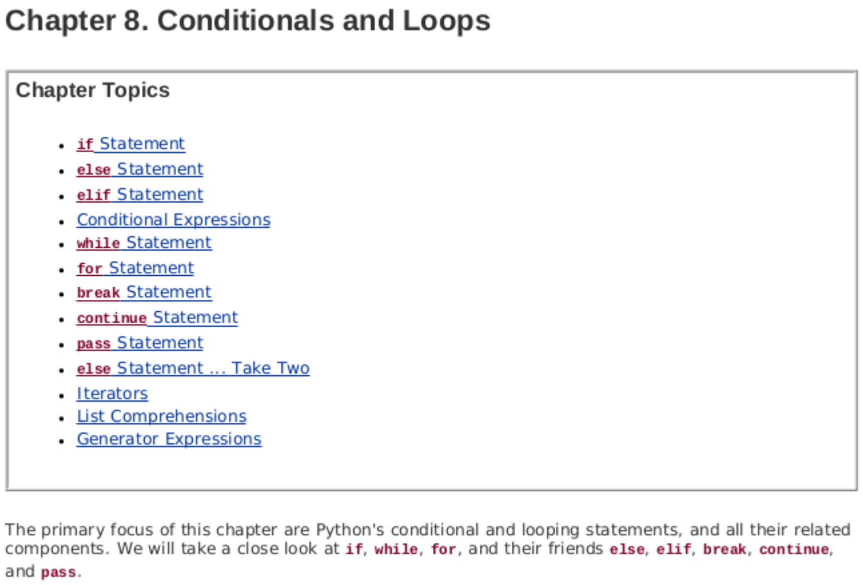


Chapter 5. Numbers

Chapter Topics

In this chapter, we will focus on Python's numeric types. We will cover each type in detail, then present the various operators and built-in functions that can be used with numbers. We conclude this chapter by introducing some of the standard library modules that deal with numbers.

Python: Conditionals and Loops

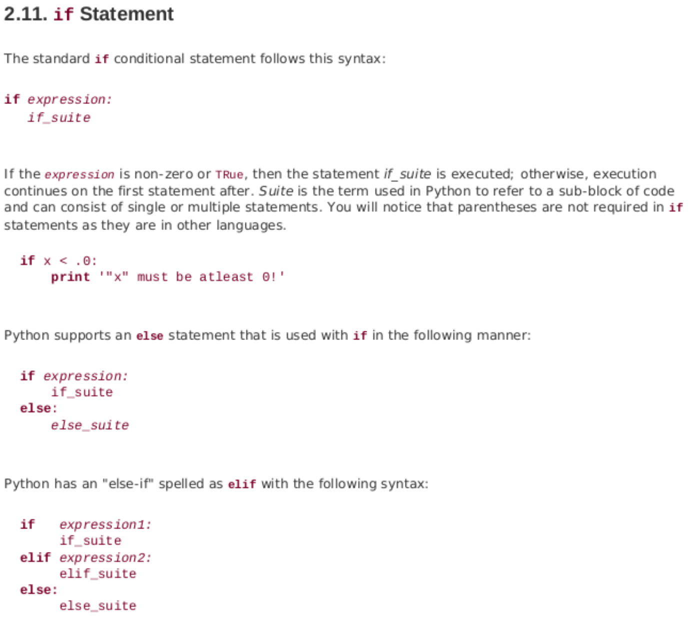


Chapter 8. Conditionals and Loops

Chapter Topics

The primary focus of this chapter are Python's conditional and looping statements, and all their related components. We will take a close look at **`if`**, **`while`**, **`for`**, and their friends **`else`**, **`elif`**, **`break`**, **`continue`**, and **`pass`**.

Conditionals: if



2.11. if Statement

The standard **if** conditional statement follows this syntax:

```
if expression:
   if_suite
```

If the *expression* is non-zero or TRue, then the statement *if_suite* is executed; otherwise, execution continues on the first statement after. *Suite* is the term used in Python to refer to a sub-block of code and can consist of single or multiple statements. You will notice that parentheses are not required in **if** statements as they are in other languages.

```
if x < .0:
    print '"x" must be atleast 0!'
```

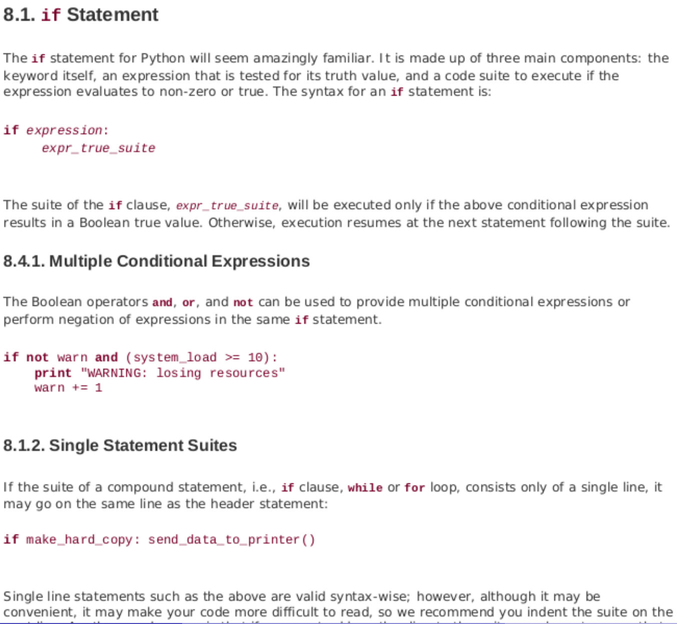
Python supports an **else** statement that is used with **if** in the following manner:

```
if expression:
    if_suite
else:
    else_suite
```

Python has an "else-if" spelled as **elif** with the following syntax:

```
if   expression1:
     if_suite
elif expression2:
     elif_suite
else:
     else_suite
```

Conditionals: if -2



8.1. **if** Statement

The **if** statement for Python will seem amazingly familiar. It is made up of three main components: the keyword itself, an expression that is tested for its truth value, and a code suite to execute if the expression evaluates to non-zero or true. The syntax for an **if** statement is:

```
if expression:
    expr_true_suite
```

The suite of the **if** clause, *expr_true_suite*, will be executed only if the above conditional expression results in a Boolean true value. Otherwise, execution resumes at the next statement following the suite.

8.4.1. Multiple Conditional Expressions

The Boolean operators **and**, **or**, and **not** can be used to provide multiple conditional expressions or perform negation of expressions in the same **if** statement.

```
if not warn and (system_load >= 10):
    print "WARNING: losing resources"
    warn += 1
```

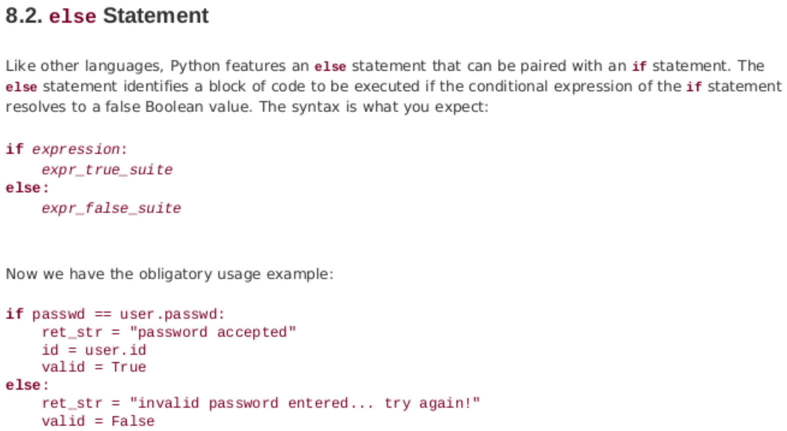
8.1.2. Single Statement Suites

If the suite of a compound statement, i.e., **if** clause, **while** or **for** loop, consists only of a single line, it may go on the same line as the header statement:

```
if make_hard_copy: send_data_to_printer()
```

Single line statements such as the above are valid syntax-wise; however, although it may be convenient, it may make your code more difficult to read, so we recommend you indent the suite on the

Conditionals: if -3



8.2. else Statement

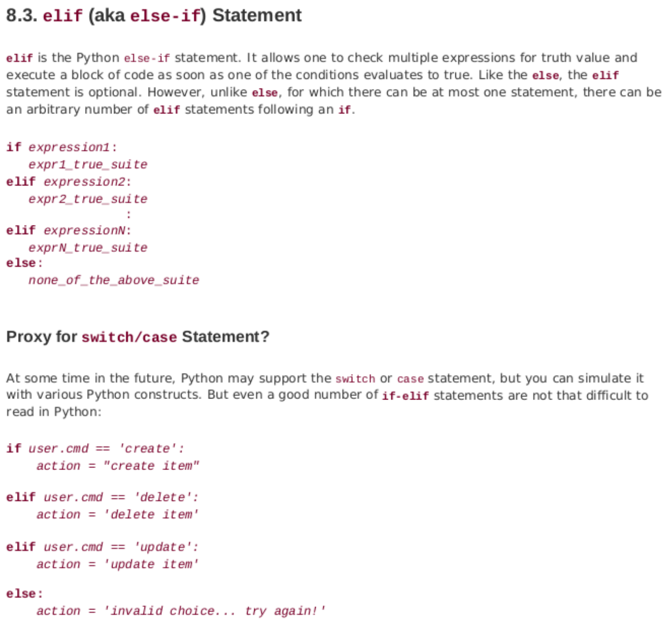
Like other languages, Python features an **else** statement that can be paired with an **if** statement. The **else** statement identifies a block of code to be executed if the conditional expression of the **if** statement resolves to a false Boolean value. The syntax is what you expect:

```
if expression:
    expr_true_suite
else:
    expr_false_suite
```

Now we have the obligatory usage example:

```
if passwd == user.passwd:
    ret_str = "password accepted"
    id = user.id
    valid = True
else:
    ret_str = "invalid password entered... try again!"
    valid = False
```

Conditionals: if -4



8.3. `elif` (aka `else-if`) Statement

`elif` is the Python `else-if` statement. It allows one to check multiple expressions for truth value and execute a block of code as soon as one of the conditions evaluates to true. Like the `else`, the `elif` statement is optional. However, unlike `else`, for which there can be at most one statement, there can be an arbitrary number of `elif` statements following an `if`.

```
if expression1:
   expr1_true_suite
elif expression2:
   expr2_true_suite
                :
elif expressionN:
   exprN_true_suite
else:
   none_of_the_above_suite
```

Proxy for `switch/case` Statement?

At some time in the future, Python may support the `switch` or `case` statement, but you can simulate it with various Python constructs. But even a good number of `if-elif` statements are not that difficult to read in Python:

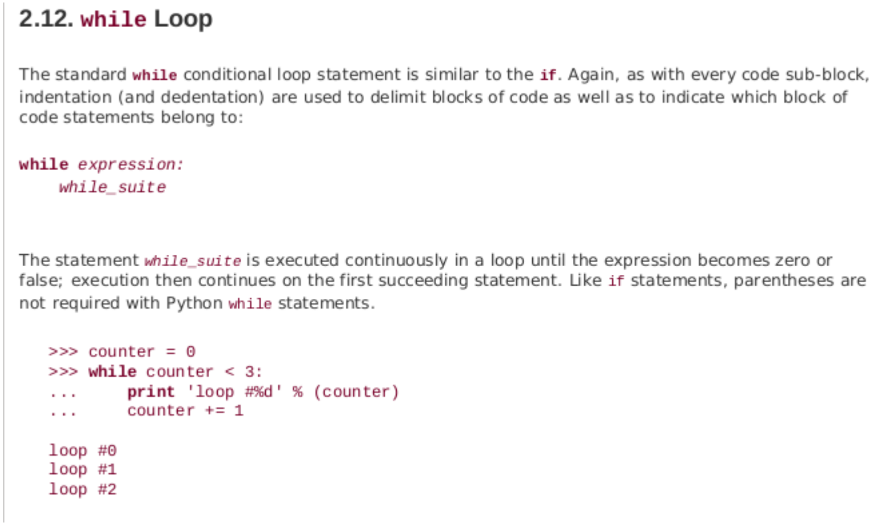
```
if user.cmd == 'create':
    action = "create item"

elif user.cmd == 'delete':
    action = 'delete item'

elif user.cmd == 'update':
    action = 'update item'

else:
    action = 'invalid choice... try again!'
```

Conditionals: while



2.12. while Loop

The standard **while** conditional loop statement is similar to the **if**. Again, as with every code sub-block, indentation (and dedentation) are used to delimit blocks of code as well as to indicate which block of code statements belong to:

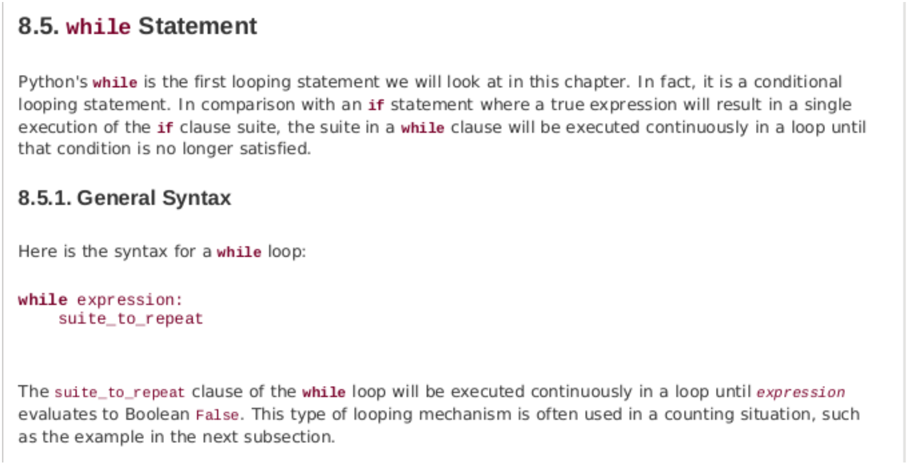
```
while expression:
    while_suite
```

The statement *while_suite* is executed continuously in a loop until the expression becomes zero or false; execution then continues on the first succeeding statement. Like if statements, parentheses are not required with Python while statements.

```
>>> counter = 0
>>> while counter < 3:
...     print 'loop #%d' % (counter)
...     counter += 1

loop #0
loop #1
loop #2
```

Conditionals: while -2



8.5. while Statement

Python's **while** is the first looping statement we will look at in this chapter. In fact, it is a conditional looping statement. In comparison with an **if** statement where a true expression will result in a single execution of the **if** clause suite, the suite in a **while** clause will be executed continuously in a loop until that condition is no longer satisfied.

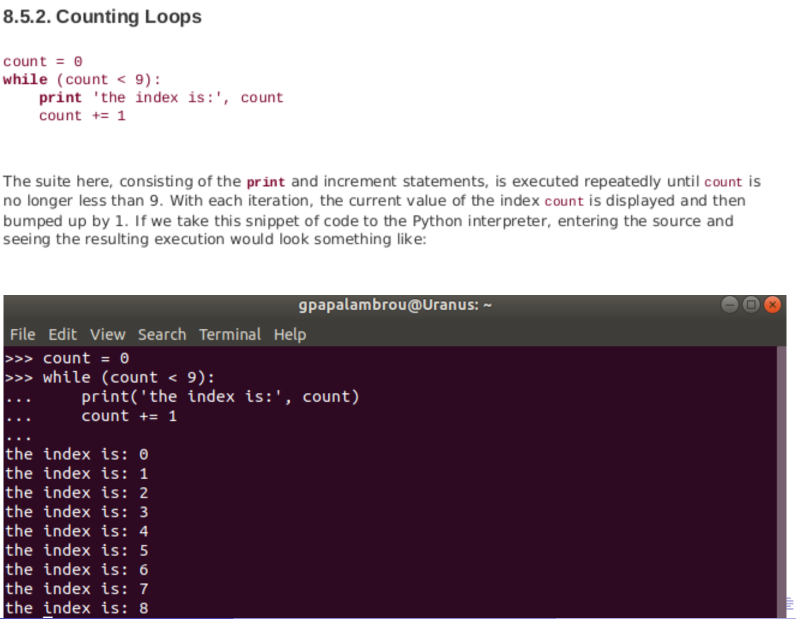
8.5.1. General Syntax

Here is the syntax for a **while** loop:

```
while expression:
    suite_to_repeat
```

The `suite_to_repeat` clause of the **while** loop will be executed continuously in a loop until *`expression`* evaluates to Boolean `False`. This type of looping mechanism is often used in a counting situation, such as the example in the next subsection.

Conditionals: while -2



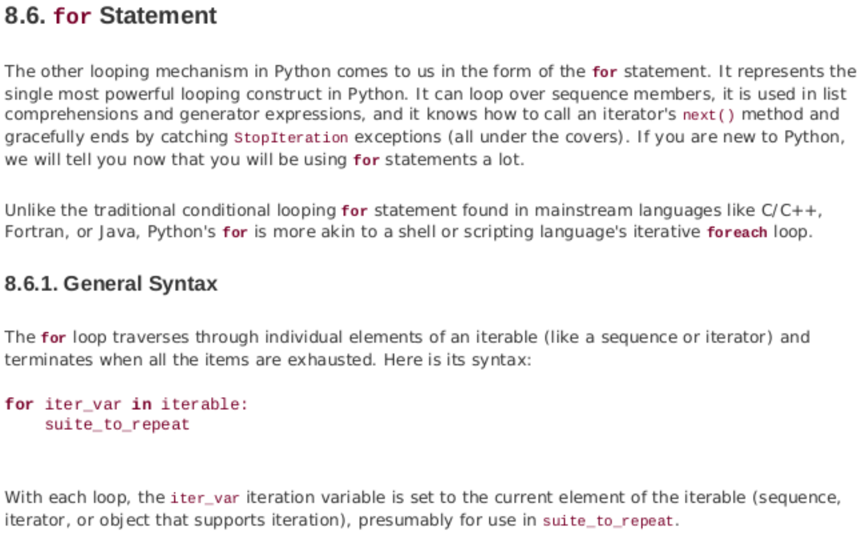
8.5.2. Counting Loops

```
count = 0
while (count < 9):
    print 'the index is:', count
    count += 1
```

The suite here, consisting of the **print** and increment statements, is executed repeatedly until count is no longer less than 9. With each iteration, the current value of the index count is displayed and then bumped up by 1. If we take this snippet of code to the Python interpreter, entering the source and seeing the resulting execution would look something like:

```
>>> count = 0
>>> while (count < 9):
...     print('the index is:', count)
...     count += 1
...
the index is: 0
the index is: 1
the index is: 2
the index is: 3
the index is: 4
the index is: 5
the index is: 6
the index is: 7
the index is: 8
```

Loops: for



8.6. for Statement

The other looping mechanism in Python comes to us in the form of the **for** statement. It represents the single most powerful looping construct in Python. It can loop over sequence members, it is used in list comprehensions and generator expressions, and it knows how to call an iterator's `next()` method and gracefully ends by catching `StopIteration` exceptions (all under the covers). If you are new to Python, we will tell you now that you will be using **for** statements a lot.

Unlike the traditional conditional looping **for** statement found in mainstream languages like C/C++, Fortran, or Java, Python's **for** is more akin to a shell or scripting language's iterative **foreach** loop.

8.6.1. General Syntax

The **for** loop traverses through individual elements of an iterable (like a sequence or iterator) and terminates when all the items are exhausted. Here is its syntax:

```
for iter_var in iterable:
    suite_to_repeat
```

With each loop, the `iter_var` iteration variable is set to the current element of the iterable (sequence, iterator, or object that supports iteration), presumably for use in `suite_to_repeat`.

Βρόχος Επανάληψης (for)

- Μπορούμε να κάνουμε την δήλωση `for` της Python σαν έναν παραδοσιακό (αριθμητικό) βρόχο και να χειριστούμε την ακολουθία ώστε να είναι μια λίστα με αριθμούς (το ⟹ δηλώνει TAB SPACE)

```
 >>> for eachNum in [0, 1, 2]:
...  ⟹ print(eachNum)
...
0
1
2
```

Βρόχος Επανάληψης (for)

- Καθώς το εύρος των αριθμών ποικίλει, η δήλωση `range()` δημιουργεί λίστα

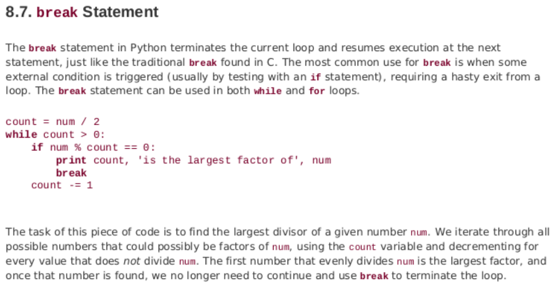
```
 >>> for eachNum in range(3):
...  ⟹ print(eachNum)
...
0
1
2
```

Βρόχος Επανάληψης (for)

- Ο τρόπος για να χρησιμοποιήσετε με προγραμματισμό έναν βρόχο για να συνθέσετε μια ολόκληρη λίστα σε μία γραμμή είναι

```
 >>> squared = [x ** 2 for x in range(4)]
>>> for i in squared:
...  ⟹ print(i)
...
0
1
4
9
```

Loops: break



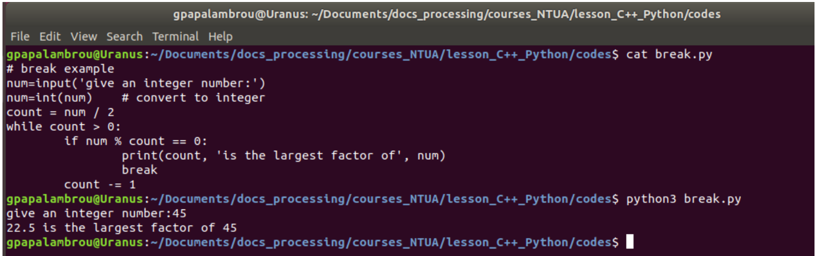
8.7. break Statement

The **break** statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional **break** found in C. The most common use for **break** is when some external condition is triggered (usually by testing with an **if** statement), requiring a hasty exit from a loop. The **break** statement can be used in both **while** and **for** loops.

```
count = num / 2
while count > 0:
    if num % count == 0:
        print count, 'is the largest factor of', num
        break
    count -= 1
```

The task of this piece of code is to find the largest divisor of a given number num. We iterate through all possible numbers that could possibly be factors of num, using the count variable and decrementing for every value that does *not* divide num. The first number that evenly divides num is the largest factor, and once that number is found, we no longer need to continue and use **break** to terminate the loop.

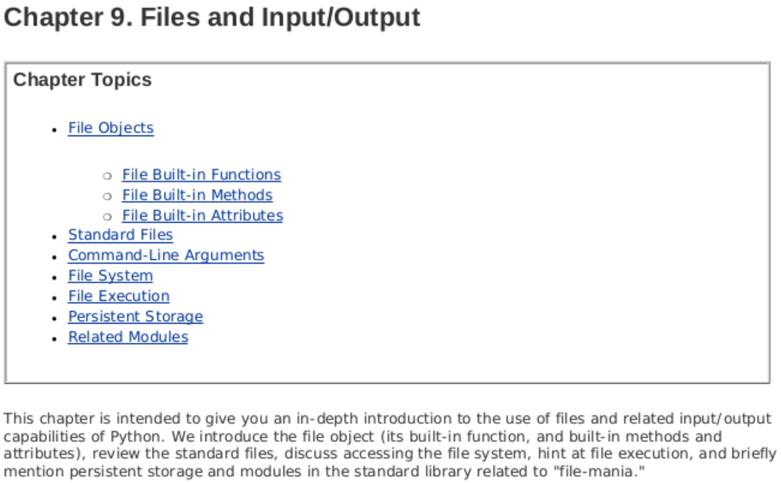
Αρχείο: break.py [GitHub]



gpapalambrou@Uranus: ~/Documents/docs_processing/courses_NTUA/lesson_C++_Python/codes
File Edit View Search Terminal Help

```
gpapalambrou@Uranus:~/Documents/docs_processing/courses_NTUA/lesson_C++_Python/codes$ cat break.py
# break example
num=input('give an integer number:')
num=int(num)    # convert to integer
count = num / 2
while count > 0:
        if num % count == 0:
                print(count, 'is the largest factor of', num)
                break
        count -= 1
gpapalambrou@Uranus:~/Documents/docs_processing/courses_NTUA/lesson_C++_Python/codes$ python3 break.py
give an integer number:45
22.5 is the largest factor of 45
gpapalambrou@Uranus:~/Documents/docs_processing/courses_NTUA/lesson_C++_Python/codes$ 
```

Python: Files



Chapter 9. Files and Input/Output

Chapter Topics

This chapter is intended to give you an in-depth introduction to the use of files and related input/output capabilities of Python. We introduce the file object (its built-in function, and built-in methods and attributes), review the standard files, discuss accessing the file system, hint at file execution, and briefly mention persistent storage and modules in the standard library related to "file-mania."

File I/O

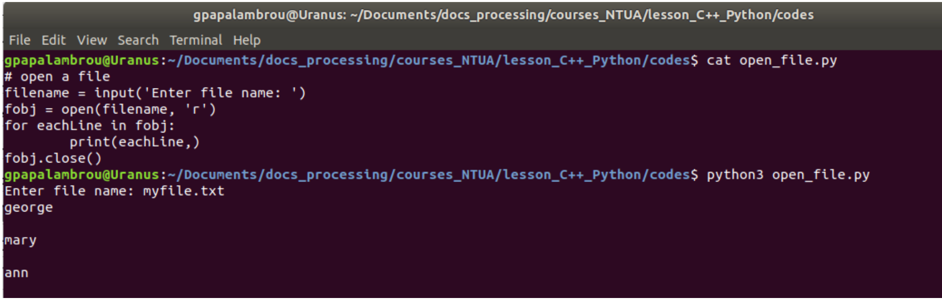
- Η πρόσβαση σε αρχεία είναι μια από τις πιο σημαντικές πτυχές της γλώσσας.
- Η δύναμη της μόνιμης αποθήκευσης είναι απαραίτητη στις πραγματικές εφαρμογές.
- Τρόπος για να ανοίξετε ένα αρχείο:
  `handle = open(file_name, access_mode = 'r')`
- Η μεταβλητή file_name περιέχει το όνομα χαρακτήρων του αρχείου που επιθυμούμε να ανοίξουμε και το access_mode είναι είτε 'r' για ανάγνωση, 'w' για εγγραφή, ή 'a' για προσάρτηση (EOF).
- Άλλα flags που μπορούν να χρησιμοποιηθούν στην access_mode περιλαμβάνουν το "+" για διπλή πρόσβαση ανάγνωσης και εγγραφής και το "b" για δυαδική (binary) πρόσβαση.
- Εάν η λειτουργία δεν παρέχεται, η προεπιλογή του μόνο για ανάγνωση ('r') χρησιμοποιείται για να ανοίξει το αρχείο.

File I/O

- Αν το `open()` είναι επιτυχές, ένα ”αντικείμενο αρχείου” (file object) θα επιστραφεί ως `handle`.
- Όλες οι επιτυχίες πρόσβασης σε αυτό το αρχείο θα πρέπει να συσχετισθούν με αυτό το handle.
- Μόλις επιστραφεί το χαρακτηριστικό του αρχείου, τότε έχουμε πρόσβαση σε όλη την λειτουργικότητα μέσω των μεθόδων όπως `readlines()`, `close()`.

File I/O

- Ο παρακάτω κώδικας (αρχείο open_file.py) [GitHub] καλεί τον χρήστη για το όνομα ενός αρχείου κειμένου (text file), ανοίγει το αρχείο και εμφανίζει το περιεχόμενο αρχείου στην οθόνη.
- Πιο πριν έχει δημιουργηθεί το αρχείο `myfile.txt` [GitHub] με περιεχόμενο 3 ονόματα

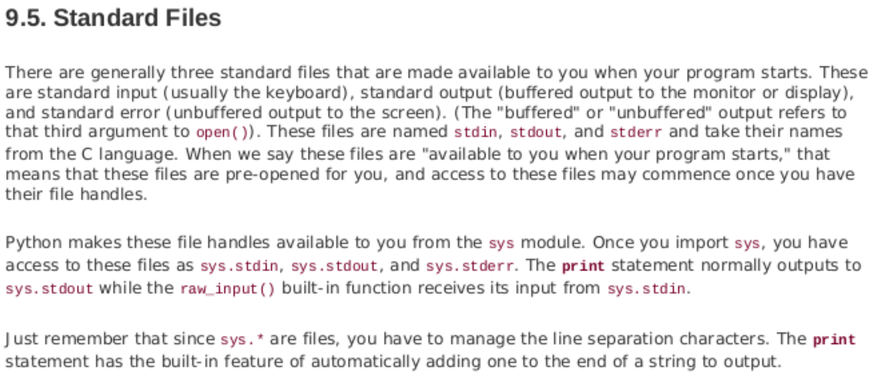

gpapalambrou@Uranus: ~/Documents/docs_processing/courses_NTUA/lesson_C++_Python/codes
File Edit View Search Terminal Help

```
gpapalambrou@Uranus:~/Documents/docs_processing/courses_NTUA/lesson_C++_Python/codes$ cat open_file.py
# open a file
filename = input('Enter file name: ')
fobj = open(filename, 'r')
for eachLine in fobj:
        print(eachLine,)
fobj.close()
gpapalambrou@Uranus:~/Documents/docs_processing/courses_NTUA/lesson_C++_Python/codes$ python3 open_file.py
Enter file name: myfile.txt
george

mary

ann
```

File - Standard Files



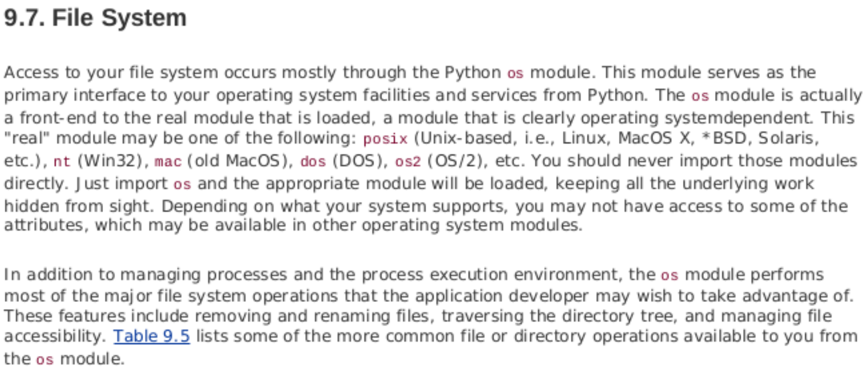
9.5. Standard Files

There are generally three standard files that are made available to you when your program starts. These are standard input (usually the keyboard), standard output (buffered output to the monitor or display), and standard error (unbuffered output to the screen). (The "buffered" or "unbuffered" output refers to that third argument to `open()`). These files are named `stdin`, `stdout`, and `stderr` and take their names from the C language. When we say these files are "available to you when your program starts," that means that these files are pre-opened for you, and access to these files may commence once you have their file handles.

Python makes these file handles available to you from the `sys` module. Once you import `sys`, you have access to these files as `sys.stdin`, `sys.stdout`, and `sys.stderr`. The **`print`** statement normally outputs to `sys.stdout` while the `raw_input()` built-in function receives its input from `sys.stdin`.

Just remember that since `sys.*` are files, you have to manage the line separation characters. The **`print`** statement has the built-in feature of automatically adding one to the end of a string to output.

File System

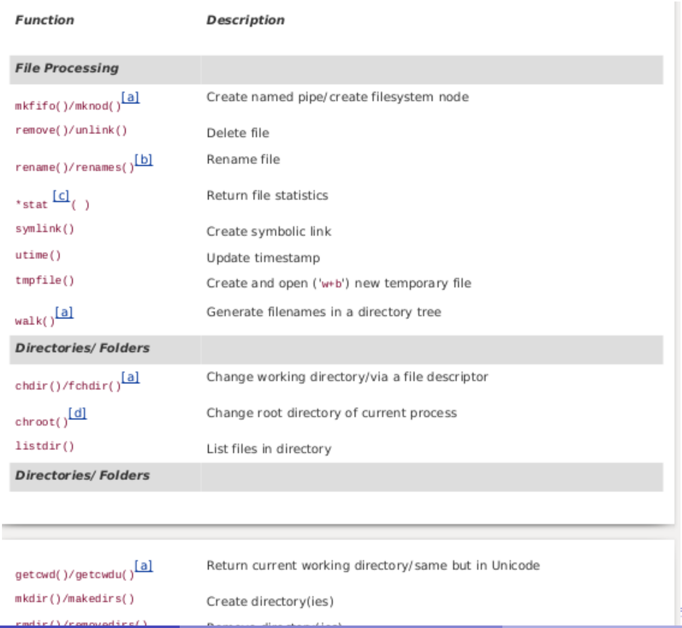


9.7. File System

Access to your file system occurs mostly through the Python os module. This module serves as the primary interface to your operating system facilities and services from Python. The os module is actually a front-end to the real module that is loaded, a module that is clearly operating systemdependent. This "real" module may be one of the following: posix (Unix-based, i.e., Linux, MacOS X, *BSD, Solaris, etc.), nt (Win32), mac (old MacOS), dos (DOS), os2 (OS/2), etc. You should never import those modules directly. Just import os and the appropriate module will be loaded, keeping all the underlying work hidden from sight. Depending on what your system supports, you may not have access to some of the attributes, which may be available in other operating system modules.

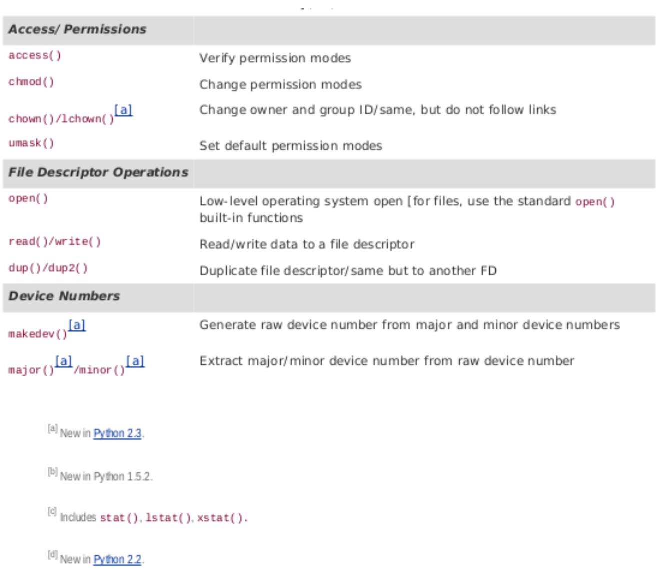
In addition to managing processes and the process execution environment, the os module performs most of the major file system operations that the application developer may wish to take advantage of. These features include removing and renaming files, traversing the directory tree, and managing file accessibility. Table 9.5 lists some of the more common file or directory operations available to you from the os module.

File System



Function	Description
File Processing	
mkfifo()/mknod()[a]	Create named pipe/create filesystem node
remove()/unlink()	Delete file
rename()/renames()[b]	Rename file
*stat [c]( )	Return file statistics
symlink()	Create symbolic link
utime()	Update timestamp
tmpfile()	Create and open ('w+b') new temporary file
walk()[a]	Generate filenames in a directory tree
Directories/ Folders	
chdir()/fchdir()[a]	Change working directory/via a file descriptor
chroot()[d]	Change root directory of current process
listdir()	List files in directory
Directories/ Folders	
getcwd()/getcwdu()[a]	Return current working directory/same but in Unicode
mkdir()/makedirs()	Create directory(ies)
rmdir()/removedirs()	Remove directory(ies)

File System



Access/ Permissions	
`access()`	Verify permission modes
`chmod()`	Change permission modes
`chown()/lchown()`[a]	Change owner and group ID/same, but do not follow links
`umask()`	Set default permission modes
File Descriptor Operations	
`open()`	Low-level operating system open [for files, use the standard `open()` built-in functions
`read()/write()`	Read/write data to a file descriptor
`dup()/dup2()`	Duplicate file descriptor/same but to another FD
Device Numbers	
`makedev()`[a]	Generate raw device number from major and minor device numbers
`major()`[a]`/minor()`[a]	Extract major/minor device number from raw device number

[a] New in Python 2.3.

[b] New in Python 1.5.2.

[c] Includes `stat()`, `lstat()`, `xstat()`.

[d] New in Python 2.2.