Examples of small Python Scripts

Back to Main Page

```
Here is a set of small scripts, which demonstrate some features of
Python programming.
```

```
# this is the first comment
```

```
#! python

# integer variables
SPAM = 1
```

```
#! python

print "Hello, Python"
```

```
#! python
# string variable

STRING = "# This is not a comment."

print STRING
```

```
#! python

# integer arith

a=4
print a

b=12+5
print b

c=b%a
print c
```

```
#! python

# trailing comma

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

```
#! python

# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1
while b < 200:
       print b,
       a, b = b, a+b
```

```
#! python

# input and operator if

x = int(raw_input("Please enter an integer: "))

if x < 0:
      x = 0
      print 'Negative changed to zero'
elif x == 0:
      print 'Zero'
elif x == 1:
      print 'Single'
else:
      print 'More'
```

```
#! python

# operator for:

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

```
#! python

# range function

print range(10)

print range(5, 10)

print range(0, 10, 3)


a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
    print i, a[i]
```

```
#! python

# break operator
# prime numbers

for n in range(2, 1000):
    for x in range(2, n):
        if n % x == 0:
            print n, 'equals', x, '*', n/x
            break
    else:
        # loop fell through without finding a factor
        print n, 'is a prime number'
```

```
#! python


#pass statement does nothing.
#It can be used when a statement is required syntactically but the program requires no action. For example:
```

```
while True:
      pass # Busy-wait for keyboard interrupt
```

```
#! python

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

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

```
! python

# function that returns a list of the numbers of the Fibonacci series

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

#===================================
f100 = fib2(100)    # call it
print f100          # write the result
```

```
#! python
# work with strings

# Strings can be concatenated (glued together) with the + operator, and repeated with *:


word = 'Help' + 'A'
print word

print '<' + word*5 + '>'

# Two string literals next to each other are automatically concatenated;
# the first line above could also have been written "word = 'Help' 'A'";
# this only works with two literals, not with arbitrary string expressions:

st='str' 'ing'             #  <-  This is ok
print st
st='str'.strip() + 'ing'   #  <-  This is ok
print st

# Strings can be subscripted (indexed); like in C, the first character of a string
# has subscript (index) 0. There is no separate character type; a character is
# simply a string of size one. Like in Icon, substrings can be specified with
# the slice notation: two indices separated by a colon.

print word[4]

print word[0:2]

print word[2:4]

# Slice indices have useful defaults; an omitted first index defaults to zero,
# an omitted second index defaults to the size of the string being sliced.

print word[:2]    # The first two characters
```

```
print word[2:]    # All but the first two characters

# Python strings cannot be changed. Assigning to an indexed position in the string results in an error:
# However, creating a new string with the combined content is easy and efficient:

print 'x' + word[1:]

print 'Splat' + word[4]

# Here's a useful invariant of slice operations: s[:i] + s[i:] equals s.

print word[:2] + word[2:]

print word[:3] + word[3:]


# Degenerate slice indices are handled gracefully: an index that is too large is replaced
# by the string size, an upper bound smaller than the lower bound returns an empty string.

print word[1:100]

print word[10:]

print word[2:1]


# Indices may be negative numbers, to start counting from the right. For example:


print word[-1]     # The last character

print word[-2]     # The last-but-one character

print word[-2:]    # The last two characters

print word[:-2]    # All but the last two characters


# But note that -0 is really the same as 0, so it does not count from the right!

print word[-0]     # (since -0 equals 0)

# Out-of-range negative slice indices are truncated, but don't try this for single-element (non-slice) indices:

print word[-100:]

# print word[-10]    # error

#The best way to remember how slices work is to think of the indices as pointing between characters,
#with the left edge of the first character numbered 0. Then the right edge of the last character
#of a string of n characters has index n, for example:

# +---+---+---+---+---+
# | H | e | l | p | A |
# +---+---+---+---+---+
# 0   1   2   3   4   5
#-5  -4  -3  -2  -1


s = 'supercalifragilisticexpialidocious'
print s
print len(s)
```

```
#! python

# Default Argument Values

def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'): return True
```

```
        if ok in ('n', 'no', 'nop', 'nope'): return False
        retries = retries - 1
        if retries < 0: raise IOError, 'refusenik user'
        print complaint

#=================================================================

i = 5

def f(arg=i):
    print arg

i = 6
f()
#=================================================================

z=ask_ok('really quit???')

if z==False :
        print "bad"
```

```
#! python

# Lambda Forms

def make_incrementor(n):
    return lambda x: x + n

#==================================
f = make_incrementor(42)

print f(0)
print f(1)
print f(15)

//=============================================================================
//
//=============================================================================
#! python

# speed test
nn=10000000
i=0;
s=0;

print "beginning..."

while i
#! python

# raw input of strings only!

st = raw_input("")

print st

st=st*3  # triple the string
print st
```

```
#! python

# math

import math

print math.cos(math.pi / 4.0)

print math.log(1024, 2)
```

```
#! python

# random

import random

print random.choice(['apple', 'pear', 'banana'])

print random.sample(xrange(100), 10)   # sampling without replacement

print random.random()    # random float

print random.randrange(6)    # random integer chosen from range(6)
```

```
#! python

def perm(l):
        # Compute the list of all permutations of l
    if len(l) <= 1:
                  return [l]
    r = []  # here is new list with all permutations!
    for i in range(len(l)):
             s = l[:i] + l[i+1:]
             p = perm(s)
             for x in p:
              r.append(l[i:i+1] + x)
    return r

#=============================================
a=[1,2,3]

print perm(a)
```

```
#! python

a=2+3j
b=2-3j

print a*a
print a*b

print a.real
print b.imag
```

```
#! python

while True:
    try:
        x = int(raw_input("Please enter a number: "))
        break
    except ValueError:
        print "Oops!  That was no valid number.  Try again..."
```

```
#! python

import string, sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(string.strip(s))
except IOError, (errno, strerror):
    print "I/O error(%s): %s" % (errno, strerror)
```

```
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
```

```
#! python
# work with lists

a = ['spam', 'eggs', 100, 1234]
print " list a=",a

# list indices start at 0,
print 'a[0]=', a[0]

print 'a[3]=', a[3]
print 'a[-2]=', a[-2]

# lists can be sliced, concatenated and so on:
print  "a[1:-1]=", a[1:-1]

print a[:2] + ['bacon', 2*2]

print 3*a[:3] + ['Boe!']

# possible to change individual elements of a list:

a[2] = a[2] + 23
print "changing a[2]=", a

#Assignment to slices is also possible, and this can even change the size of the list:

# Replace some items:
a[0:2] = [1, 12]
print a

# Remove some:
a[0:2] = []
print a

# Insert some:
a[1:1] = ['bletch', 'xyzzy']
print a

a[:0] = a     # Insert (a copy of) itself at the beginning
print a

print "length=", len(a)

# possible to nest lists (create lists containing other lists)

q = [2, 3]
p = [1, q, 4]
print " nest list=", p

print 'length =', len(p)

print p[1]

print p[1][0]

p[1].append('xtra')
print p

print q
```

```
#! python

# more work with lists
```

```
a = [66.6, 333, 333, 1, 1234.5]

print a.count(333), a.count(66.6), a.count('x')

a.insert(2, -1)
print a

a.append(333)
print a

print a.index(333)

a.remove(333)
print a

a.reverse()
print a

a.sort()
print a
```

```
#! python

# huge list making

nn=1000000
a = []
i=0

while i
```

```
#! python

# Using Lists as Stacks

stack = [3, 4, 5]
stack.append(6)
stack.append(7)
print stack

x=stack.pop()
print "popped ",x
print stack

x=stack.pop()
print "popped ",x
x=stack.pop()
print "popped ",x
print stack
```

```
#! python

# Using Lists as Queues

queue = ["Eric", "John", "Michael"]
queue.append("Terry")           # Terry arrives
queue.append("Graham")          # Graham arrives
print queue

s=queue.pop(0)
print s

s=queue.pop(0)
print s

print queue
```

```
#! python

# The del statement

a = [-1, 1, 66.6, 333, 333, 1234.5]
del a[0]
print a

del a[2:4]
print a
```

```
#! python

# filter of sequence

def f(x): return x % 2 != 0 and x % 3 != 0

res=filter(f, range(2, 25))

print res
```

```
#! python

# map of sequence

def cube(x): return x*x*x

res=map(cube, range(1, 11))

print res
```

```
#! python

# reduce(func, sequence)" returns a single value constructed by
# calling the binary function func on the first two items of the sequence,
# then on the result and the next item, and so on

def add(x,y): return x+y

r=reduce(add, range(1, 11))
print r # 55
```

```
#! python

# A tuple consists of a number of values separated by commas

t = 12345, 54321, 'hello!' # tuple packing
print t[0]

print t
(12345, 54321, 'hello!')

# Tuples may be nested:
u = t, (1, 2, 3, 4, 5)
print u     #  ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

```
#! python

# Dictionaries are sometimes as ``associative memories'' or ``associative arrays''

tel = {'jack': 4098, 'sape': 4139}
```

```
tel['guido'] = 4127
print tel

print tel['jack']

del tel['sape']
tel['irv'] = 4127
print tel

print tel.keys()

x=tel.has_key('guido')
print x

# The dict() constructor builds dictionaries directly from lists
# of key-value pairs stored as tuples. When the pairs form a pattern,
# list comprehensions can compactly specify the key-value list.

d=dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
print d

vec=[1,2,3,4,5]
dd=dict([(x, x**2) for x in vec])     # use a list comprehension
print dd
```

```
#! python

# Standard Module sys

import sys

print sys.path

sys.path.append('c:\temp')
print sys.path

print sys.version
print sys.platform

print sys.maxint
```

```
#! python
#=========================================================
# dir() is used to find out which names a module defines

import sys

print dir(sys)

# Without arguments, dir() lists the names you have defined currently
```

```
#! python

#  convert any value to a string: pass it to the repr() or str()

s = 'Hello, world.'
print str(s)

print repr(s)

print str(0.1)

print repr(0.1)

x = 10 * 3.25
y = 200 * 200
s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'
```

```
print s

# The repr() of a string adds string quotes and backslashes:
hello = 'hello, world\n'
hellos = repr(hello)
print hellos   # 'hello, world\n'

# The argument to repr() may be any Python object:
print repr((x, y, ('spam', 'eggs')))

# reverse quotes are convenient in interactive sessions:
print `x, y, ('spam', 'eggs')`
```

```
#! python

#  two ways to write a table of squares and cubes:


for x in range(1, 11):
    print repr(x).rjust(2), repr(x*x).rjust(3),
    # Note trailing comma on previous line
    print repr(x*x*x).rjust(4)


print '==============================================='
for x in range(1,11):
    print '%2d %3d %4d' % (x, x*x, x*x*x)
```

```
#! python

# output results from running "python demo.py one two three"
# at the command line:

import sys
print sys.argv[]  #  ['demo.py', 'one', 'two', 'three']
```

```
#! python

# String Pattern Matching - regular expression

import re

r=re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
print r # ['foot', 'fell', 'fastest']

s=re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
print s # 'cat in the hat'
```

```
#! python

# dates are easily constructed and formatted

from datetime import date

now = date.today()
print now

datetime.date(2003, 12, 2)

print now.strftime("%m-%d-%y or %d%b %Y is a %A on the %d day of %B")
```

```
# dates support calendar arithmetic

birthday = date(1964, 7, 31)
age = now - birthday
print age.days # 14368
```

```
#! python

# Internet Access

import urllib2

for line in urllib2.urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl'):
        if 'EST' in line:      # look for Eastern Standard Time
            print line

import smtplib

server = smtplib.SMTP('localhost')
server.sendmail('soothsayer@tmp.org', 'jceasar@tmp.org',
"""To: jceasar@tmp.org
From: soothsayer@tmp.org

Beware the Ides of March.
""")
server.quit()
```

```
# work with files

#open file for write

f=open('c:/TEMP/workpy.txt','w')

print f

f.write("aaaaaaaaaaaaaaaaaa\n")
f.write("bbbbbbbbbbbbbb");
```

```
# work with files

#open file for read

f=open('c:/TEMP/workpy.txt','r')

# line reading

s=f.readline()
print s

f.close()
```

```
# work with files

#open file for read

f=open('c:/TEMP/workpy.txt','r')

# pieces reading

s1=f.read(5)
print s1
```

```
s2=f.read(19)
print s2

s2=f.read(25)
print s2

f.close()
```

```
# work with files

#open file for read

f=open('c:/TEMP/workpy.txt','r')

# pieces reading

s1=f.read(5)
print s1
print f.tell()

s2=f.read(19)
print s2
print f.tell()

s2=f.read(25)
print s2
print f.tell()

f.close()
```

```
# work with files

# seek

f=open('c:/TEMP/workpy.txt','r+')

f.write('0123456789abcdef')
f.seek(5)     # Go to the 6th byte in the file
print f.read(1)

f.seek(-3, 2) # Go to the 3rd byte before the end
print f.read(1)
```

```
#! python

# The glob module provides a function for making file lists from
# directory wildcard searches:

import glob

s=glob.glob('*.*')
print s  #  ['primes.py', 'random.py', 'quote.py']
```
