Basic Python Elements

This article is an introduction to the most basic Python operations. This information can be found in much greater detail in the Python documentation available at http://python.org/

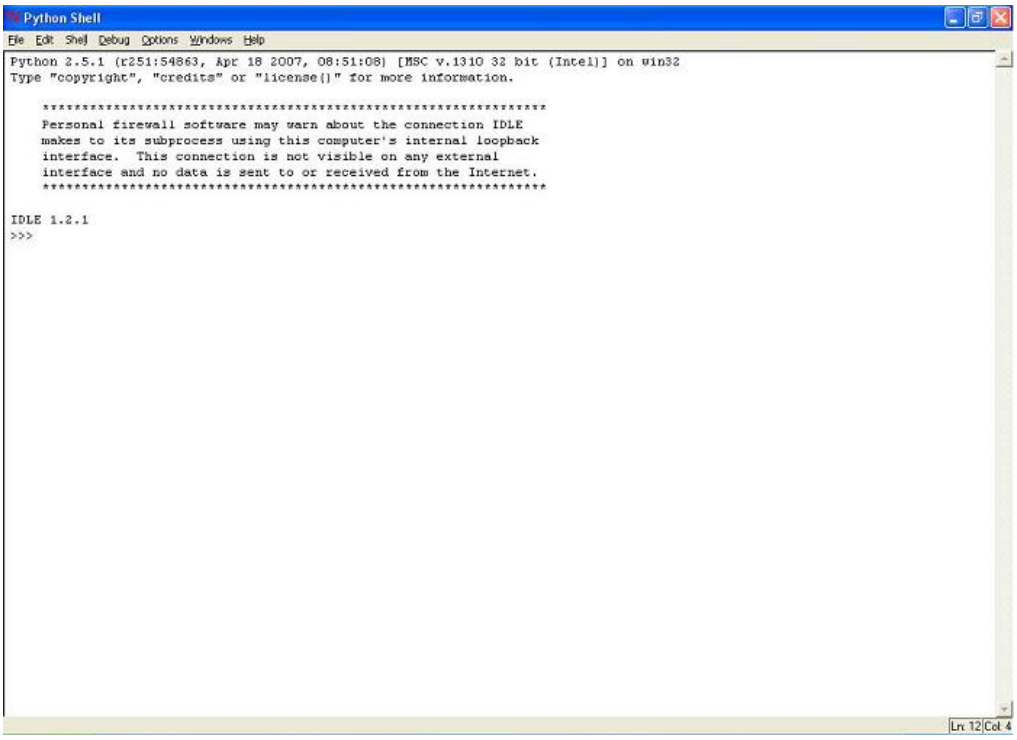
For an overview and installation instructions of PySymbian, see FNWiki Python.

12 Oct 2008

Description

Python is a programming language similar to C++, yet it is more automated. The user has to do less in order to achieve the same result (that is, the type of a variable does not have to be declared but Python analyses it and stores it accordingly). Python consists of two parts: the PC editor and compiler, used to write and test the code, and the on-phone interpreter, which reads and executes the code. The Python editor (IDLE or command line) can be used for testing applications in real time. However, some users prefer entering code in a traditional text editor, such as Notepad.

The following snippets can be entered in interactive mode. In this mode, the interpreter prompts for the next command with the primary prompt, usually three greater-than signs ">>> ". For continuation lines it prompts with the secondary prompt, "... ". If the instruction is not complete, the interpreter displays the secondary prompt.



Variables

Variables are "containers" used to store values. For example:

```
x=2
```

stores the value 2 in variable x.

The following types of variables are available in Python:

-integer - a number

-long - an extended version of integer

-float - a number with decimals

-boolean - a logical variable (can be TRUE or FALSE)

-character - a text character (like '1' or 'f')

-string - a sequence of 1 or more characters (like 'Nokia' or '2')

There are other kinds of variables called arrays of elements (strings and lists). An array is a collection of items, such as:

(C) Copyright Nokia 2013. All rights reserved.

```
list=[1,2,3,4] #a list of numbers
list=['a','b'] # a list of characters
#etc
```

An array is characterised by its length (the number of elements). For example, the length of [1,2,3,6,1,'d'] is 6. Note that an array is evaluated from position 0 (element 1, in this case) to its length-1 (element d).

```
>>> a=[1,2,3]
>>> a[2]
3

>>> s="alabala"
>>> s[3]
b
```

To check whether a short and well-known element is in an array:

```
element in array #returns true if the element is in that array, and false if not.
#Example:
>>> x=2
>>> if (x in [1,5,2,7,2,4]):
...     print 'yes'
... else:
...     print 'no'
...
yes
```

After entering the ":" symbol, press the 0 key 3 times (for entering the return key). The secondary prompt is displayed because the "if" statement is not completed. Enter now 4 spaces before entering the next line. Repeat these steps until the statement is fully completed.

A dictionary is a built-in data structure represented by a pair of elements (key and value) that allows you to access items according to their keys rather than their position in the dictionary.

```
>>> d = {"saturday":"theatre", "sunday":"lunch"}
>>> print d["sunday"]
lunch
```

Variables can be local or global. Local variables are the ones defined inside a function and whose scope goes not go beyond that function. Global variables, on the other hand, are known during the entire execution of the code. A local variable x can be made global using the statement "global x".

Program syntax and other general information

The input is what the user tells the program.

x=2, x=raw_input() are ways to give x a value.

Output is what the program shows the user.

print x outputs the value of x.

You can also use print "%letter" % (value). The letters and associated types are:

f - float

d - integer

s - string

Example:

```
>>> print "%d" % (5)
5
>>> print "%s" % ('S60')
S60
>>> x=2.14
>>> print "%f" % (x)
2.14
```

"#" in the program code starts a comment (a line that is not interpreted by Python. The programmer can use it to describe certain operations).

Operators and operations

Operators

Operators are used to compare variables. < (smaller), > (greater), <= (smaller or equal), == (equal), >= (greater or equal), != (different)

```
x=2
y=5

x<y
x<=y
x!=y
#return true

x>=y
```


(C) Copyright Nokia 2013. All rights reserved.

```
x>y
x==y
#return false
```

"and", "or", "not" - used with boolean variables (true and true=true, true and false=false, t or t=t, t or f=t, not t=f, not f=t)

```
x=2
y=3
(x<y) and (y!=0) #returns true
not(x==2) #returns false
```

General operations

Like in any programming language, the programmers can manipulate the variables in almost any way they desire. Here is a list of operations, the types of variables they can be applied to, and examples:

- "+" -integer (2+5=7), long (500000000+1000000000=6000000000), float (2.34+1.2=3.54), character ('f'+'g'='fg') and string ('nokia'+' n95'='nokia n95')

- "-" -integer, long, float

- "*" -integer, long, float, or between an integer and a character or string (3*'hi'='hihihi')

- "/" -integer, long, float

- "%" -the modulus, meaning the number you get after dividing a number to another number, long, float (2.4%2=0.4)

Special operations

These operations apply to arrays.

-Changing an element

```
>>> x = [1, 1, 1]
>>> x[1] = 2
>>> x
[1, 2, 1]
```

-Deleting elements

```
>>> names = ['Alice', 'Beth', 'Cecil', 'Dee-Dee', 'Earl']
>>> del names[2]
>>> names
['Alice', 'Beth', 'Dee-Dee', 'Earl']
```

-Separating elements

```
>>> name = list('Perl')
>>> name
['P', 'e', 'r', 'l']
```

-Appending

```
>>> lst = [1, 2, 3]
>>> lst.append(4)
>>> lst
[1, 2, 3, 4]
```

Conditionals and loops

In order to check if a condition is true or not, the 'if' structure is used. Based on the validity of the condition, instructions can be given.

```
if <condition1>:
    <instruction>
    .
    .
elif <condition2>:
    <instruction>
    .
    .
else:
    <instruction>
    .
    .
```

The condition1 is checked. If it is true, the instructions under it will be carried out. If condition1 is false, condition2 is checked. If it is true, the instructions under it will be carried out. If, on the other hand, neither condition1 or condition2 are true, the instructions under the 'else' part will be carried out. The 'elif' part can be omitted for simpler conditional structures.

Example:

```
>>> x=2
```


(C) Copyright Nokia 2013. All rights reserved.

```
>>> if(x==2):
...     print 'ok'
...
ok

>>> if (x==56):
...     print '56'
... else:
...     print x
...
2

>>> a = 3
>>> b = 4
>>> if (a % 2 == 0):
...     print 'even a'
... elif (b % 2 == 0):
...     print 'even b'
... else:
...     print 'something else'
...
even b
```

When an action needs to be repeated for a certain number of times or until a condition is met, use the 'while' or 'for' loop:

```
while <condition>:
    <instruction1>
    <instruction2>
```

If the condition is true, the loop is entered and the instructions will be performed until the condition becomes false. Example:

```
>>> i=1
>>> while (i<=4):
...     print 'ok'
...     i+=1
...
ok
ok
ok
ok
```

```
>>> for i in range(3):
...     print "do"
...
do
do
do
```

Functions

Functions are used to perform a set of operations to modify a certain element or for many other purposes. An example of a function is the "pow" function:

```
>>> x=2
>>> y=3
>>> pow(x,y)
8
```

A function is declared as follows:

```
def functionname(parameters):
    <operations>
    return result
```

Functions can also be used to convert a type of variable into another. The structure is variable2=function(variable1) These functions are: int(var) and long(var)->number, float(var)->number with decimals, str(var)->string, chr(var)->character. Example:

```
>>> x=2.34
>>> print int(x)
2
>>> str(x)
'2.34'
```

When used in functions, variables can be local (modifications do not apply to their value outside the function) or global (after the function is performed, the value will be modified). See the following example:

```
>>> x=2
>>> def inc(x):x+=1
>>> x
2
>>> inc(x)
>>> x
2
#notice that after calling the function inc, the value of x remains unchanged
```

Compare that with:

```
>>> x=2
>>> def inc():
...     global x
...     x+=1
...
>>> x
2
>>> inc()
```


(C) Copyright Nokia 2013. All rights reserved.

```
>>> x
3
#here, if the value of x is modified in the function, this modification is kept for the rest of the program
```

Importing modules

Modules contain certain functions that are not loaded with Python by default, but instead need to be loaded by the user. There are two methods for importing modules or extensions:

```
>>> import math
>>> sin(3.14159)
0
#or
>>> from math import sin
>>> sin(3.14159)
0
```

The difference is that the first method imports the entire "math" module, while the second only imports the needed function, "sin".

An important module for PySymbian is "appuifw". It contains the functions used to get certain types of input and output. Example:

```
import appuifw
appuifw.note(u"This is an example")
# will give you a message containing the text "This is an example"
```

Other important modules are "graphics" and "e32".

Developing applications

Python is one of the easiest languages for developing applications for mobile phones and other platforms. Let's say we want to make an application that asks you to pick a name from a list and displays the code associated with it. Here is a step-by-step guide to a method:

1.Create a new a Notepad document.

2.Write the code of the application.

```
#import the necessary modules
import appuifw
import e32

#give the program a title
appuifw.app.title=u'Database'
#display the list of names so that the user can pick one. The order is important
pickedname=appuifw.popup_menu([u"Martha", u"Gregory"])
#associate a code to each of them
codes=[1206,2367]
#display the corresponding code, given by the number of the picked name from the list
if(pickedname==0):appuifw.popup_menu([u"%d" % (codes[0])])
if(pickedname==1):appuifw.popup_menu([u"%d" % (codes[1])])
```

3.Save the file as filename.py. The extension is important.

4.Transfer file to your phone. Store it in the "Python" folder.

5.Test it. Open Python, select Option->Run script->filename.py.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.