

Python Quick Fact Sheet

By: Mike Angstadt (dim_vision@hotmail.com)

Last Updated: 25 October 2004

Note: Some examples were taken directly from Python's tutorial

Website: <http://www.python.org>

"Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on PCs under MS-DOS, Windows, Windows NT, and OS/2."

-Python FAQ

General

- # use for comments
- use `and`, `or`, `not` instead of C's `&&`, `||`, `!`
- `**` is used for raising to the power
 - ex) `3**4` returns 81
- code blocks: a colon and indentation are used instead of brackets like with C
 - code blocks cannot be empty
 - ex) `if x == 5:`
 `→ print x`

Variables and Data Structures

- created on the fly, no declarations
- `del varName` - deletes variable
- `global varName` - gives a variable global scope
- Strings:
 - **cannot** be edited once declared
 - can be declared with double or single quotes
 - individual characters can be accessed like in C
 - `r"hello\nworld"` returns `"hello\nworld"`
 - *given*: `s = "gghellogg"`:
 - `s.strip("gg")` returns `"hello"`
 - `len(s)` returns 9
 - `s[3]` returns `"e"`
 - `s[3] = "f"` - not a legal command
 - `del s[3]` - not a legal command
 - `s[1:4]` returns `"ghe"`
 - `s[2:]` returns `"hellogg"`
 - `s[:2]` returns `"gg"`
 - `s[:-1]` returns `"gghellog"`

- `"start" + s + "end"` returns `"startgghelloggend"`
- `s*2` returns `"gghellogggghellogg"`

- Lists:

- basically a dynamic array
- can have multiple data types in the same list
- can use element retrieval like with strings
 - ex) `listName[1:3]`
- can have multi-dimensional lists
- *given*: `L = ['unicorn', 42, 78.45]`
 - `L + [2]` returns `['unicorn', 42, 78.45, 2]`
 - `del L[1]` would change L to `['unicorn', 78.45]`
 - `len(L)` returns 3
- methods:
 - `append(var)` - adds var to the end of the list
 - `extend(listVar)` - adds the list listVar to the end of list
 - the same as `list1 + list2`
 - `insert(index, value)` - inserts value at the index position
 - `remove(v)` - removes the first item in the list with value v, returns error if no item is found
 - `pop()` - returns then removes the last item
 - `pop(index)` - returns then removes the index element
 - `index(v)` - returns the index of the first item with value v, returns error if no item is found
 - `count(v)` - returns number of time value v is in the list
 - `sort()` - sorts list in ascending order
 - `reverse()` - reverses the list
- used for stacks: `L.pop()`
- used for queues: `L.pop(0)`
- list comprehensions:
 - always returns a new list; never modifies the original
 - *given*: `a = [1, 2, 3, 4]`
 - `[item+1 for item in a if item > 2]` returns `[1, 2, 4, 5]`
 - `[[b, b**2] for b in a]` returns `[[1,1], [2,4], [3,9], [4, 16]]`
- performs vector arithmetic:
 - *given*: `a = [1,2,3]; b = [4,5,6]`
 - `[x*y for x in a for y in b]` returns `[4, 5, 6, 8, 10, 12, 12, 15, 18]`

-Dictionaries

- *given*: `d = {'ripcord': 29, 'ansaphone': 43}`
 - `d['ripcord']` returns 29
 - `d['airbag'] = 37` adds an entry to d, modifying d to `{ 'ripcord' : 29, 'ansaphone': 43, 'airbag': 37}`
 - `del d['airbag']` deletes that entry
 - prints out the contents:

```

for x, y in d.items():
    print x, y

```

- methods:

- keys() – returns a list of the keys (the strings) in the dictionary
- has_key(keyStr) - returns 0 or 1 if keyStr is in the dictionary
- items() – returns a 2-d array of the keys and their values

Loops and If Statements

- continue continues with the next iteration of the loop
- break breaks the current loop
- pass does nothing (good for debugging, since each loop needs *something* in its body)
- else: code to execute when a loop is terminated normally (if it doesn't use break)
 - included right after the loop
 - its contents must be indented

- For

- to print the numbers from 1 to 9:

```

for x in range(1,10):
    print x

```

same as C's:

```

for (x = 1; x < 10; x++)
    printf("%d\n", x);

```

- to print the contents of the entire list:

```

for x in aList:
    print x

```

same as C's:

```

for (x = 0; x < aListLen; x++)
    printf("%d\n", aList[x]);

```

- While

```

while x < 5 or y >= 6:

```

- If/Else

```

if x == 5 and y == 6:
    #code
elif s == "yourmom":
    #code
else:
    #code

```

Functions

- syntax:


```

def theFunc(var1, var2, var3):

```
- del theFunc – deletes a function
- return keyword is used to return a value
- returns the value None if nothing else is returned
- can “rename” them:


```

f = theFunc

```

 - f can now be called just as theFunc is called

- default arguments:

```

def theFunc(var1, var2 = 'blah', var3 = 42)

```

- can pass in variables in any order as long as the variable name is specified

```

theFunc(var2 = 'myxomatosis', var1 = 6)

```

- for multiple arguments:

```

def multi(*args):
    for x in args: print x
multi("raindrops", "scatterbrain", 87)

```

- for dictionaries:

```

def diction(**keywords):
    keys = keywords.keys()
    keys.sort()
    for kw in keys: print kw, ":", keywords[kw]
diction(pig='fitter', cage='happier', anti-biotics='productive')

```

- documentation

```

def theFunc(var1, varN):
    """This does nothing to when the function is called"""
    #function body

```

- returns that string when theFunc.__doc__ is called

Useful Functions

** variables in brackets indicate an optional parameter and default value*

- chr(char) – returns the ascii value of the character
- dir() – returns a list of all defined variables and functions
- filter(funcName, range(...))
 - funcName – the name of a function which must have exactly one parameter
 - returns a list of the values (which are specified by range(...)) that were passed into funcName which caused funcName to return true
- map(funcName, range(...))
 - like filter(...) but returns all values returned by funcName in a list
- print
 - print(varName) – prints varName to the screen
 - print val1, val2, val3 - prints multiple variables each separated by a space
- range([start = 0], end, [inc = 1]) - used in loops and if statements


```

if x in range(1, 10):
    #body

```

same as C's:

```

if (x >= 1 && x < 10)
    //body

```
- str(varName) – returns a stringed version of varName
- zip(list1, [list2], [listN]) - used to print the contents of two or more lists in a loop


```

for x, y in zip(list1, list2):
    print x, y

```

same as C's:

```

for (x = 0; x < length; x++)
    printf("%d %d\n", list1[x], list2[x]);

```