

Python 2.7.x

© 2014-07-31 Martin Bruchanov, bruxy@regnet.cz

Set of interpreter: `#!/usr/bin/env python`

Comments: `# everything behind hash`

`""" more lines comment """`

Command line parameters

- `python options script.py` – run script filename
- `-V` – print version
- `-c 'code'` – run code from command line

2. Expression statements

FOR cycle	WHILE contition
<pre>for identifier in list : list-processing code [else : suite]</pre>	<pre>while condition repeat if condition is true [else: suite]</pre>
IF-THEN-ELSE	TRY block
<pre>if condition : true suite [elif condition: else if true] [else : else suite]</pre>	<pre>try: possible runtime error except [type [as value]]: error-recovery code [else: suite] [finally: suite]</pre>

- `import module` – find and initialize *module*
- `module.function()` – use function of imported module
- `from module import *` – import all stuff to local name space
- `import module as name` – rename imported module
- `from module import name as othename`
- `break` – exit while or for loop, skip associated else
- `continue` – perform next iteration of cycle
- `global name` – reference global value
- `exec("print 'Ahoj'")` – compile and exec code
- with *expression* [as *variable*]:
 - suite* – block entry actions
- `pass` – do-nothing placeholder statement
- `del name, del name[i], del name[i:j:k], del name.attribute` – delete variables, items, keys, attributes
- `assert expression [, message]`
- `exec codestring`

2.1. Classes

- `class Name:`
 - suite*
- `_private` – underscored named object is private
- `def __init__(self, ...):`
 - `self.data = []` – constructor
- `class DerivedClass(BaseClass)` – inheritance
- `def __iter__(self):` –

2.2. Functions

- `def function(param1, param2,...):`
 - `pass`
- `def func(arg,... arg=value, ... *arg, **arg):`
 - `arg` – matched by name or position
 - `arg=value` – default value if arg is not passed
 - `*arg` – collect extra positional args as a new tuple
 - `**arg` – collect extra positional args as a new dictionary
- `lambda args1 : expression` – anonymous function maker
- `return [expression]` – return from function
- `yield expression` – suspend function state and return, on next iteration restore prior state

3. Variables

- `variable = 12` – assign value
- `type(variable)` – return type of variable
- `global name [,name]` – global variable in local context
- Number formats:**
 - `2006, 20061, 2006L` – decimal integer, long;
 - `0775, oct(0x1fd)` – octal;
 - `0xBABE, hex(47806)` – hexadecimal;
 - `0b101010, bin(42)` – binary;
 - `3.14, 314e-2` – floating point;
 - `1+2j, 1.0+2.0J, complex(1,2)` – complex number;
 - `b'Ahoj'` – sequence of 8-bit values;
- `int(x), long(x), float(x), str(n)` – type conversions

- `c=1+2j; c.conjugate(), (1+2j).conjugate()` – conjugate of complex number $1 - 2j$
- `abs(x)` – absolute value of *x*
- `round(x[,n])` – *x* rounded to *n* digits
- `(10.5).as_integer_ratio()` – returns tuple (21, 2)
- `(255).bit_length()` – number of digits of binary
- `X, Y = Y, X` – swap values of *X* and *Y*
- `a,b,c = range(3)` – read list values, `a=0,b=1,c=2`
- `vars()` – return dictionary of variables and values
- `globals(), locals()` – return dictionary of variables
- `setattr(obj, 'b', c)` is equivalent `obj.b = c`
- `getattr(obj, 'a')` is equivalent `obj.a`
- `hasattr(obj, name)` – True if name is object attribute

3.1. Constants

- `False, True` – boolean
- `None` – represents no value
- `bool([X])` – returns boolean value of object *X*.

4. Operators

- `or, and, not x` – boolean operators
- `| (or), ^ (xor), & (and), ~x (neg.)` – binary operators
- `X in Y, X not in Y` – membership tests
- `X is Y, X is not Y` – same or different object
- `<, <=, >, >=, <>, !=, ==` – comparisons
- `*, /, //, %` – multiply, divide, floor divide, remainder
- `x << n, x >> n` – bitwise shifts by *n* bits
- `x**y, pow(x,y)` – power x^y
- `+= &%= -= |= *%= ^= /= >>= \%= <<= **= //=`
- `divmod(x,y)` – return tuple (*x/y, x%y*)

5. Data types

Function	Tuple	List	Dict.	String	Set
Init.	<code>()</code> , <code>tuple()</code>	<code>[]</code> , <code>list()</code>	<code>{}</code> , <code>dict()</code>	<code>""</code> , <code>'</code> , <code>_str()</code>	<code>set()</code>
<code>clear</code>	—	—	•	—	•
<code>copy</code>	—	—	•	—	•
<code>count</code>	•	•	—	•	—
<code>index</code>	•	•	—	•	—
<code>pop</code>	—	•	•	—	•
<code>remove</code>	—	•	—	—	•
<code>update</code>	—	—	•	—	•

5.1. Tuples

- `t = (), t = tuple()` – create empty tuple
- `t = (1, 2, 3)` – like list, but can't change their values
- `t[1]` – access second item, returns 2
- `t.index(x [, i [, j]])` – return index of first occurrence of *x*
- `t.count(x)` – return number of item *x*

5.2. Lists

- `l = [], l = list()` – empty list
- `l = [1, 2, 3]` – one dimensional array
- `l[1]` – returns 2, indexing: $l_0 \ 2_1 \ 3_2$
- `l[i:j]` – slicing from index *i* to *j*
- `l[i:]` – slicing from index *i* to end of list
- `l[i:j:k]` – slicing with step $k \approx 1$ [`slice(i, j [,k])`]
- `l[-1]` – last item (first from back)
- `0 in [1, 2, 3]` – False, `1 in [1, 2, 3]` True
- `l = range(5)` – create list [1, 2, 3, 4, 5]
- `l = range(start, stop[, step])` – given range with step
- `l = [x**2 for x in range(9)]` – list from expression result
- `l.index(item)` – return index of *item* in list
- `l.count(item)` – total number of occurrences of *item*
- `l = ["text", 12, 3, [1, 2]]` – more types in one list
- `l2d=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]` – two-dimensional list
- `l2d[1][1]` – returns 5
- `list('abc')` – returns list of chars ['a', 'b', 'c']
- `len(l)` – return length of list
- `l.append(value)` – add *value* to the list
- `l.extend([4,5]), list[len(list)]=[4,5], list += [4,5]` – append another list
- `l.insert(i, x), list[i]=x` – insert *x* at given index
- `l[:0]=[x,y,z]` – insert item at front of list
- `l.remove(value)` – remove first occurrence of value
- `l.pop(i), l.pop()` – return and remove value, without index last
- `l.index(x [, i [, j]])` – index of first occur. of *x*, between *i* to *j*
- `l.count(x)` – return number of occurrence of object *x*
- `l.sort(key=None, reverse=False)` – sort list in-place
- `l.reverse()` – reverse list in-place
- `sum(l)` – return sum of numeric list

5.3. Dictionaries

- `h = {}, h = dict()` – initialization of empty dictionary
- `h = {"key1": "value", "key2": "another"}` – definition
- `h = dict(key1="value", key2="another")` – different syntax
- `h["key3"] = 333` – add another value
- `h = {c: ord(c) for c in 'spam'}` – comprehension expression
- `h.has_key("key")` – returns True if key exist
- `h.keys()` – return list of keys
- `h.values()` – return list of values
- `h.clear()` – remove all items
- `g = h.copy()` – returns a shallow copy of *h*
- `h.get(key [, default])` – if key is not found return *default*
- `h.popitem()` – removes and returns an (*key, value*) pair
- `h.pop(k [, def])` – returns and removes *k* else return *def*
- `h.fromkeys(seq [, value])` – new dictionary from keys in *seq*
- `dict(zip(['a', 'b'], [1,2]))` – join to {'a': 1, 'b': 2}

5.4. Sets

- `A = set()` – empty set $A = \{\emptyset\}$
- `A = set('Ouagadougou')` – $A = \{'a', 'd', 'g', 'o', 'u', 'O'\}$, unordered collection of unique and immutable objects
- `A = {'a', 'd', 'g', 'o', 'u', 'O'}` – set definition
- `A = frozenset(range(-5, 5))` – immutable set of $-5...4$
- `'a' in A` – returns True if value is presented $a \in A$
- `A - B, A.difference(B)` – new set contains difference $A \setminus B$
- `A | B, A.union(B)` – join two sets, no duplicates $A \cup B$
- `A & B, A.intersection(B)` – same items in both sets $A \cap B$
- `A <= B, A.issubset(B)` – returns True is *A* is subset of *B* $A \subset B$
- `A >= B, A.issuperset(B)` – is *A* superset of *B*? $A \supset B$
- `A < B, A > B` – true subset, superset $A \subset B, A \subset B$
- `A ^ B, A.symmetric_difference(B)` – $A \triangle B = (A \cup B) \setminus (A \cap B)$
- `A |= B, A.update(B)` – adds item in *B* to *A*
- `A.discard(X)` – remove item if exist
- `A.add(X), A.remove(X)` – add, remove item from set
- `A.clear()` – remove all items
- `A.pop()` – remove and return arbitrary item
- `len(A)` – get number of items in *A*
- `for x in A:` – all iteration context
- `B=A.copy(), B=set(A)` – make copy of set

5.5. Strings

- `s = "Hello", s = 'Hello'` – definition, " and ' works same
- `"""This is multi-line block"""` – collects into a single string
- `s[1]='e'` – indexing $H_0 \ e_1 \ l_2 \ l_3 \ o_4$
- `str(n)` – convert number *n* to string
- `'Hello ' + 'World', "Hello" "World"` – concatenation
- `'Hello' * 3` – repetition $3 \times$
- Unicode α : `u"\03b1", U"\U000003B1", u"\N{GREEK SMALL LETTER ALPHA}"`
- Raw string: `r"\n", R'\n'` does not interpret escape sequences
- Unicode raw string: `ur"\n", UR'\n'`
- `str(), bytes(), bytearray()` – create string from object
- `\xhh, \ooo, \0` – hex, octal, null byte
- `chr(65), unichr(65), ord('A')` – returns character, ASCII code
- `eval(s)` – convert and execute code given by string
- `execfile(filename)` – like eval, but for whole file

6. Output and formatting

- `print(*objects, sep=' ', end='\n', file=sys.stdout)`
- `'%s, %s, %.2f' % (13, 'txt', 22/7.0)` – '13, txt, 3.14'
- `'{0}, {1}, {2:.2f}'.format(13, 'txt', 22/7.0)` – other def.
- `%(a)d %(b)s" % {"a":6, "b": "text"}` – formatting dictionary
- `"{a} {b}".format(**{'a':1, 'b':2})` – formatting dictionary
- `"%s" % (10, "text")` – width given as parameter
- `"%#x %#o" % (15,15)` – prints number base prefixes
- `"%.*f" % (5, 22.0/7)` – +3.14286, 5 digits after "
- `%[(keyname)][flags][width][.precision]typecode`
- Flags: `-/+` left/right justify, `0/'` ' zero/space fill
- String formatting typecodes:**
 - `s` – String (or any object, uses `str()`)
 - `r, - s`, but uses `repr()`, not `str()`
 - `c` – Character (int or str)
 - `d, i, u` – Decimal (base 10 integer)
 - `o` – Octal (base 8 integer)
 - `x, X` – Hex (base 16 integer)
 - `e, E` – Floating-point exponent
 - `f, F` – Floating-point decimal
 - `g, G` – Floating-point `e, f/E, f`
 - `%` – Literal `'%`
- `{fieldname!conversionflag:formatspec}`

- `[[fill]align [sign] [#] [0] [width] [,] [.prec] [typecode]`

7. String methods

- `s.find/rfind(sub, [,s [,e]])` – index of first occur. of sub,
- `s.index/rindex(sub [,s [,e]])` – ValueError if not found
- `s.endswith/startwith(sub [,s [,e]])` – true if starts/ends
- `s.count(sub, [,s [,e]])` – get number of substrings
- `s.upper(), s.lower(), s.swapcase()` – converts case
- `s.split([sep [, maxsplit]])` – return list of words
- `sep.join(iterable)` – concatenates with separator
- `' and '.join(['a', 'b', 'c'])` – returns 'a and b and c'
- `s.replace(old, new [, count])` – replace *old* by *new*
- `s.splitlines(0/1)` – split by `\n`, 1 – keeps end char
- `s.strip([chars])` – remove leading and trailing white spaces
- `s.lstrip, s.rstrip` – just from left or right side
- `s.center/ljust/rjust(width [,fill])` – justify string
- `s.capitalize() / s.title()` – make first/all word(s) uppercase
- `s.expandtabs(tabsize)` – replaces tabs with spaces (default 8)
- `isalnum, isalpha, isdecimal, isdigit, isidentifier, islower, isnumeric, isprintable, isspace, istitle, isupper` – tests

8. Other build-in functions

- `max(iterable), min(iterable)` – return max/min value
- `reversed(iterable)` – return a reverse iterator
- `sorted(iterable, key=None, reverse=False)` – return sorted
- `enumerate(iterable, start=0)` – return an enumerate object
- `all(iter), any(iter)` – True if all/any of elements are/is true.
- `hash(obj)` – return hash value of object
- `iter(o [,sentinel])` – return an iterator object
- `next(iterator [,default])` – return next item from the iterator
- `map(function, iterable, ...)` – apply function on every item
- `input([prompt])` – read line for stdin

9. Work with files

- `file = open('data.txt'), mode` – open file, mode: `r, w, rb, w, r+, w+`
- `s = file.read([n])` – read file of *n* bytes into string *s*
- `file.readline()` – return line of file, empty at EOF
- `file.readlines()` – read entire file into a list of line strings
- `for line in file:` – process file line by line
- `file.write(s)` – write string *s* into file
- `file.writeline(list)` – write all string in list to file
- `print >>file, "Output"` – write string to file
- `file.close()` – close to free resources
- `file.tell()` – return file position
- `file.seek(offset [, whence])` – set file position
- `file.flush()` – flushes file's buffer
- `file.truncate([size])` – truncate file to size bytes
- `file.fileno()` – get file descriptor integer
- `file.closed, file.mode, file.name` – return attributes

10. Regular expressions (import re)

- `ro=re.compile(pattern, flags=0)` – create *RegexObject* „ro“
- Flags: `re.DOTALL (S)`, `re.IGNORECASE (I)`, `re.LOCALE (L)`, `re.MULTILINE (M)`, `re.VERBOSE (X)`, `re.UNICODE (U)`
- `re.match(pattern, string)` – if match return *MatchObject*
- `re.search(pattern, string)` – match regex anywhere in string
- `re.split(pattern, string)` – split pattern
- `re.findall(pattern, string)` – return substrings as list
- `re.finditer(pattern, string)` – return matches as iterator
- `re.sub(pattern, repl, string, count=0, flags=0)` – return string with replaced pattern
- `re.subn(...)` – return tuple (string, num. of replacements)
- `re.escape(string)` – string with escaped regex's metacharacters
- RegexObject methods:** `ro.match, search, split, sub, subn, findall, finditer`
- `ro.flags, ro.pattern` – used argument for reg. obj. creation
- `ro.groups()` – number of matched groups
- `ro.group(n)` – return *n*th string matched by regex
- `ro.start(), ro.end(), ro.span()` – return starting, ending position or tuple

11. System specific functions and parameters

- `sys.argv` – CLI parameters, `argv[0]` name of script
- `sys.stdin.readline()` – read line from standard input
- `subprocess.call(["ls", "-l"])` – execute system command
- `out = subprocess.check_output(['uname', '-a'])` – store output of command to variable
- `filelist = subprocess.Popen("ls *", shell=True, stdout=subprocess.PIPE).communicate()[0]` – read data from pipe