Python

© 2021 Martin 'BruXy' Bruchanov, bruchy@gmail.com

Set of interpreter: `#!/usr/bin/env python`
Comments: `# everything behind hash`
`""" more lines comment """`

1. Command line parameters

- `python` *options* `script.py` – run script filename
- `-V` – print version
- `-c '`*code*`'` – run code from command line
- `python -m py_compile` *script.py* – test syntax of script
- `python3 -c 'import keyword; print(keyword.kwlist)'` – keywords
- `python -i`, `ipython` – interactive mode

2. Create virtual environment

- `python -m virtualenv /path/to/dir` or `python3 -m venv`
- Make current shell to use it: `source /path/to/dir/bin/activate`
- Check if virtual env. is used: `pip --version`
- Quit virtual env.: `deactivate`

3. Expression statements

FOR cycle	WHILE contition

`for` *identifier* `in` *list* `:`
`  list-processing code`
`[ else :`
`  suite ]`
	`while` *condition*
	`  repeat if condition is true`
	`[ else:`
	`  suite ]`

IF-THEN-ELSE	**TRY block**

`if` *condition* `:`
`  true suite`
`[ elif` *condition*`:`
`  else if true ]`
`[ else :`
`  else suite ]`
	`try:`
	`  possible runtime error`
	`except [type [as value]]:`
	`  error-recovery code`
	`[ else:`
	`  suite ]`
	`[ finally:`
	`  suite ]`

- `import` *module* – find and initialize *module*
- *module.function()* – use function of imported module
- `from` *module* `import *` – import all to local name space
- `import` *module* `as` *name* – rename imported module
- `from` *module* `import` *name* `as` *othername*
- `break` – exit while or for loop, skip associated else
- `continue` – perform next iteration of cycle
- `quit([code=`*exit code*`])` – exit script and set return value
- `raise Exception('Message')` – create error during execution
- `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.attibute` – delete variables, items, keys, attributes
- `assert` *expression* `[,` *message*`]`
- `exec` *codestring*
- Generator expression:
  result expr. `for` *loop var.* `in` *iterable* `if` *filter expr.*
- `s = 'Yes' if k == True else 'No'` – ternary operator
- `def noop(*args, **kws): return None` – no-operation function

3.1. Classes

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

3.2. 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
- `map(lambda x: x.capitalize(), ['abc','def'])` – example
- `return [`*expression*`]` – return from function
- `yield` *expression* – suspend function state and return, on next iteration restore prior state
- `def gen(x):\n for i in range(x): yield i*i` – returns gener. object

4. Variables

- `variable = 12` – assign value
- `type(variable)` – return type of variable
- `global name [,name]` – global variable in local context
- **Number formats:**
  - `2006, 2006l, 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
- `int('GEEK', 21)` – convert string number with given base
- `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(), 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 atribute

4.1. Constants

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

5. 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)`

6. Data types

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

6.1. 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

6.2. Lists

- `l = []`, `l = list()` – empty list
- `l = [1, 2, 3]` – one dimensional array
- `[3] * 4` – repeated 4× to `[3, 3, 3, 3]`
- `l[1]` – returns 2, indexing: $1_0\ 2_1\ 3_2$
- `l[i:j]` – slicing from index i to $j-1$
- `l[i:]` – slicing from index i to end of list
- `l[i:j:k]` – slicing with step $k \approx$ `l[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 [0, 1, 2, 3, 4]
- `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 $j-1$
- `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
- `sorted(set(l))` – sort and unique list via set
- `sorted(unsorted_l, key=str.casefold)` – sort case insensitive

6.3. Dictionaries

- `h = {}`, `h = dict()` – initialization of empty dictionary
- `h = {"key1": "value", "key2": "another"}` – definition
- `h = dict(key1="value", key2="another")` – different syntax
- `h = dict([('key1', 'value'), ('key2', 'another')])` – yet another
- `h["key3"] = 333` – add another value
- `h[(1+2j)] = 666` – also hashable object can be a key
- `h = {c: ord(c) for c in 'spam'}` – comprehension expression
- *'key'* `in h` – returns `True` if key exist (was `h.has_key("key")`)
- `h.keys()`, `h.values()` – return list of keys, and 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*
- `del h['key1']` – remove entry for key *key1*
- `h.fromkeys(seq [, value])` – new dictionary from keys in *seq*
- `dict(zip(['a','b'], [1,2]))` – join to `{'a': 1, 'b': 2}`
- `for key, value in h.items():` – iterate dictionary
- `g = {}; g.update(h)`, `g = dict(h)`, `g = dict(h.items())` – make copy

6.4. Sets

- `A = set()` – empty set $A = \{\emptyset\}$
- `A = set('Ouagadougou')` = `A = set(['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 $A \subset B$, $A \supset B$
- `A ^ B`, `A.symmetric_difference(B)` – $A \triangle B = (A \cup B) \setminus (A \cap B)$
- `A |= B`, `A.update(B)` – adds items 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

6.5. Strings

- `s = "Hello"`, `s = 'Hello'` – definition, `"` and `'` works same
- `s[::-1]` – reverse string (olleH)
- `"""This is multi-line block"""` – collects into a single string
- `s[1]='e'` – indexing $H_0\ e_1\ l_2\ l_3\ o_4$
- `s[i:j]` – substring from i to $j-1$ (same indexing as lists)
- `str(n)` – convert number n to string
- `'Hello ' + 'World'`, `"Hello" "World"` – concatenation
- `'Hello' * 3` – repetition 3×
- Unicode '→': `"\u2192"`, `"\U00002192"`, `"\N{Rightwards Arrow}"`
- 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

7. Output and formating

- `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"}` – formating dictionary
- `"{a} {b}".format(**{'a':1, 'b':2})` – formating 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 formating 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]`
- `vprint = print if VERBOSE else lambda *a, **k: None` – verbose prints

8. 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/startswith(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

9. Other built-in functions

- `help(`*object*`)`, `help('`*function*`')` – display documentation
- `max(iterable)`, `min(iterable)` – return max/min value
- `reversed(iterable)` – return a reverse iterator
- `sorted(iterable, key=None, reverse=False)` – return sorted
- `sorted(lt, key=lambda x: x[1])` – sort list of tuples by 2nd element
- `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 iterator
- `map(function, iterable, ...)` – apply function on every item
- `input([`*prompt*`])` – read line for stdin

10. Work with files

- `file=open('data.txt'[, '`*mode*`'])` – open, 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
- `print >>file, "Output"` – write string to file
- `file.writeline(list)` – write all strings in list 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
- `with open('file.txt', 'r') as f:` – block with file manipulations

11. 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

12. 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
- `os.stat('`*/path/to/file.txt*`')` – return POSIX stat file info
- `os.environ.get('PATH')` – get value of environment variable `PATH`