Python 3 Quick Reference

Hubert Högl

Revision 0.4, 2018-02-12

https://github.com/huberthoegl/pqr3

Contents

E-mail: <Hubert.Hoegl@t-online.de>

ENVIRONMENT VARIABLES

PYTHONSTARTUP: file executed on interactive startup (no default)

PYTHONPATH: ':'-separated list of directories prefixed to the default module search path. The result is sys.path.

PYTHONHOME: alternate <prefix> directory (or <prefix>:<exec_prefix>). The default module search path uses <prefix>/pythonX.X.

PYTHONCASEOK: ignore case in 'import' statements (Windows).

PYTHONIOENCODING: Encoding[:errors] used for stdin/stdout/stderr.

PYTHONFAULTHANDLER: dump the Python traceback on fatal errors.

PYTHONHASHSEED: if this variable is set to 'random', a random value is used to seed the hashes of str, bytes and datetime objects. It can also be set to an integer in the range [0,4294967295] to get hash values with a predictable seed.

KEYWORDS

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

PYTHON3

```
hhoegl@e11 ~ $ python3
Python 3.5.1 |Anaconda 2.4.0 (32-bit)| (default, Dec  7 2015, 11:17:45)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

- Basic Python prompt.
- Use only for short "throw-away" interactive work.
- For comfortable interactive work use `IPython`.

IPYTHON3

```
hhoegl@e11 ~/anaconda3/bin $ ./ipython3
Python 3.5.1 |Anaconda 2.4.0 (32-bit)| (default, Dec  7 2015, 11:17:45)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: _
```

- Get help with `?` or `??` after module, function, object, identifier.

```
import os
os?
id?
```

Press `TAB` key to auto-complete.

Enter `quickref` for a list of magic commands.

- The `%editor` command (alias `%ed`) starts the editor which is defined in the `EDITOR` environment variable.
- The IPython configuration is in directory `~/.ipython/`. More help: `ipython config -h`
- IPython help: `ipython3 --help`
- IPython Notebook help: `ipython3 notebook --help`
- ```
  ipython3 --pylab
  ipython3 --matplotlib tk
  ipython3 --matplotlib qt4
  ```

- Default coding style for Python source files is `UTF-8`. Use `# -*- coding: <encoding name> -*-` for alternate encodings.
- Can use UTF-8 characters in strings, comments and identifiers:

  ```
  müll = "tröööööt"
  ```

  Further reading (Unicode): [5]

NUMBERS

Types: Integer, Float, Complex

Numbers are immutable

```
>>> i = 12
>>> j = -20
>>> u, v, w = 10, 12, 14

>>> p = 2 ** 100         # 2^100
>>> p
1267650600228229401496703205376

>>> f = 3.1415           # double (8 byte)

>>> 2.3 ** 100
1.4886191506362924e+36

>>> c = 4 + 3j           # complex number
```

Operations on Numbers

Operation	*Description*	*Notes*
`X + Y, X - Y`	Add, subtract	
`X * Y, X / Y`	multiplication, division	
`X // Y, X % Y`	integer division, reminder	
`-X, +X`	sign	
`X | Y, X & Y`	bitwise or, bitwise and	
`X ^ Y`	exor	
`X « n, n » n`	shift	
`~X`	bitwise negate	
```

`X ** Y`	pow	
`abs(X)`	absolute value	
`int(X)`	type conversion	
`float(X)`	type conversion	
`complex(X)`	type conversion	
`divmod(X, Y)`	integer division and reminder	
`pow(X, Y, [Z])`	identical with `**`	

- Also see modules `decimal` (decimal floating point arithmetic) and `fractions` (rational number arithmetic).

Integer methods

```
i.bit_length   i.denominator  i.imag         i.real
i.conjugate    i.from_bytes   i.numerator    i.to_bytes
```

Example:

```
>>> i = 4 ** 100
>>> i.bit_length()
201
```

Float methods

```
f.as_integer_ratio  f.fromhex           f.imag              f.real
f.conjugate         f.hex               f.is_integer
```

Complex methods

```
c.conjugate  c.imag       c.real
```

ASSIGNMENTS

```
a = 42              a **= b
a += b              a &= b
a -= b              a |= b
a *= b              a ^= b
a /= b              a >>= b
a //= b             a <<= b
a %= b

a = 12 if cond else 0   # conditional expression
```

COMPARISONS

Comparisons are defined for any types.

Operation	*Description*	*Notes*
<	less than	
<=	less or equal	
>	greater than	

`>=`	greater or equal	
`==`	equal	
`!=`	not equal	
`is`	identity	
`is not`	negated identity	

- Works as expected: `0.2 <= f < 0.9`

SEQUENCES, MAPPINGS and COLLECTIONS

```
                  Sequences                          Mappings         Collections
               /             \                          |              /      \
        immutable              mutable                 mutable          mut    immut
       /    \      \           /     \                    |              |       |
 strings  bytes  tuples      list  bytearray             dict           set    frozenset


            INDEX: 0, 1, 2, 3, ...                  NO INDEX!!!
```

Lit.: `https://docs.python.org/3/reference/datamodel.html`

`Examples:`

```
>>> s = "Hello World"
>>> B = bytes([3, 2, 9, 250])
>>> B[3]
250
>>> T = (7.32, "Hallo", 12)
>>> L = [3, "Hallo", 4.71, 2+5j]
>>> BA = bytearray([4, 9, 0, 255])
>>> BA[2] = 13
>>> D = { 0 : "Null", 1 : "Eins"}

>>> T = (1, 4, [4, 5, 9], 90)
>>> T[2][1] = 99

>>> b = True
>>> c = not True

>>> v = None
```

- `bytes()` → `b'...'`

Operations on all sequences

Operation	*Description*	*Notes*
`len(S)`	Number of elements in S	
`x in S`	x member in S	
`x not in S`	x not member in S	
`S1 + S2`	Concatenation	
`S * n`	Sequence times int	
`n * S`	Int times sequence	
`S[i]`	Index Operator	

`S[i:j]`	Slice Operator	
`S[i:j:k]`	Slice Operator	
`S.index(x)`	Get index of element x	
`min(S)`	Minimum	
`max(S)`	Maximum	
`iter(S)`	Return Iterator	
`for x in S:`	Iteration	
`[e for e in S]`	List comprehension	
`map(f, S)`	Functional programming	
`filter(f, S)`	Functional programming	

Operations on mutable sequences

Operation	*Description*	*Notes*
`S[i] = x`	Assign x at index i in S	
`S[i:j] = T`	Assign T to slice	
`S[i:j:k] = T`	Assign T to slice	
`del S[i]`	Delete	
`del S[i:j]`	Delete	
`del S[i:j:k]`	Delete	

- Change L *in place*: `L[:] = [`*list compr. with L*`]`

Operations on Mappings

Operation	*Description*	*Notes*
`D[k]`	Lookup dict by key k	
`D[k] = x`	Key assignment	
`del D[k]`	delete entry	
`len(D)`	number of items in D	
`k in D`	was `D.has_key(k)`	
`k not in D`	test for key not in D	
`iter(D)`	get iterator over keys (`dict_keyiterator`)	
`for k in D:`	iterate over keys	

Operations on Sets

Operation	*Description*	*Notes*
`len(S)`	number of elements in S	
`x in S`	x is element of Set S	
`S1 - S2`	difference	
`S1 -= S2`	difference update	
`S1 | S2`	union	
`S1 |= S2`	union update	
`S1 & S2`	intersection	
`S1 ^ S2`	symmetric difference	
`S1 ^= S2`	symmetric difference update	
`S1 < S2`	test for true subset	
`S1 <= S2`	test for subset	
`S1 > S2`	test for true superset	
`S1 >= S2`	test for superset	
`S1 != S2`	test for unequal sets	

See also section SET METHODS below.

STRING METHODS

IPython Hilfe (`s.<tab>`)

```
s.capitalize    s.format        s.islower       s.lower         s.rpartition    s.title
s.casefold      s.format_map    s.isnumeric     s.lstrip        s.rsplit        s.translate
s.center        s.index         s.isprintable   s.maketrans     s.rstrip        s.upper
s.count         s.isalnum       s.isspace       s.partition     s.split         s.zfill
s.encode        s.isalpha       s.istitle       s.replace       s.splitlines
s.endswith      s.isdecimal     s.isupper       s.rfind         s.startswith
s.expandtabs    s.isdigit       s.join          s.rindex        s.strip
s.find          s.isidentifier  s.ljust         s.rjust         s.swapcase
```

Method	*Description*	*Notes*
`s.capitalize()`	Returns a copy of s with its first character capitalized, and the rest of the characters lowercase.	
`s.casefold()`	Return a version of s suitable for caseless comparisons.	
`s.center(width[, fillchar])`	Returns a copy of s centered in a string of length *width*, surrounded by the appropriate number of *fillchar* characters.	
`s.count(sub[, start[, end]])`	Return number of non-overlapping occurrences of substring sub in string s[start:end]. Optional arguments start and end are interpreted as in slice notation.	
`s.encode([encoding [, errors]])`	Encode s using the codec registered for encoding. Return `bytes`.	
`s.endswith(suffix [,start [,end]])`	Return True if s ends with the specified suffix, False otherwise.	
`s.expandtabs([tabsize])`	Return a copy of s where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 8 characters is assumed.	
`s.find(sub [,start [,end]])`	Return the lowest index in s where substring sub is found, such that sub is contained within s[start:end]. Optional arguments start and end are interpreted as in slice notation.	
`s.format(*args, **kwargs)`	see `https://pyformat.info`	
`s.format_map(mapping)`	Return a formatted version of s, using substitutions from mapping.	
`s.index(sub [,start [,end]])`	Like s.find() but raise ValueError when the substring is not found.	
`s.isalnum()`	Return True if all characters in s are alphanumeric and there is at least one character in s, False otherwise.	
`s.isalpha()`	Return True if all characters in s are alphabetic and there is at least one character in s, False otherwise.	
`s.isdecimal()`	Return True if there are only decimal characters in s, False otherwise.	
`s.isdigit()`	Return True if all characters in s are digits and there is at least one character in s, False otherwise.	
`s.isidentifier()`	Return True if s is a valid identifier according to the language definition.	
`s.islower()`	Return True if all cased characters in s are lowercase and there is at least one cased character in s, False otherwise.	
`s.isnumeric()`	Return True if there are only numeric characters in s, False otherwise.	
`s.isprintable()`	Return True if all characters in s are considered printable in repr() or s is empty, False otherwise.	
`s.isspace()`	Return True if all characters in s are whitespace and there is at least one character in s, False otherwise.	
`s.istitle()`	Return True if s is a titlecased string	
`s.isupper()`	Return True if all cased characters in s are uppercase and there is at least one cased character in s, False otherwise.	
`s.join(iterable)`	Return a string which is the concatenation of the strings in the iterable. The separator between elements is s.	

`s.ljust(width [,fillchar])`	Return s left-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).	
`s.lower()`	Return a copy of the string s converted to lowercase.	
`s.lstrip([chars])`	Return a copy of the string s with leading whitespace removed. If chars is given and not None, remove characters in chars instead.	
`s.maketrans()`	Return a translation table usable for `s.translate()`.	
`s.partition(sep)`	→ (head, sep, tail). Search for the separator sep in s, and return the part before it, the separator itself, and the part after it. If the separator is not found, return s and two empty strings.	
`s.replace(old, new [,count])`	Return a copy of s with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.	
`s.rfind(sub [,start [,len]])`	Return highest index in s where substring sub is found, such that sub is contained within s[start:end].	
`s.rindex(sub [,start [,len]])`	Like s.rfind() but raise ValueError when the substring is not found.	
`s.rjust(width [,fillchar])`	Return s right-justified in a string of length width. Padding is done using the specified fill character (default is a space).	
`s.rpartition(sep)`	Search for the separator sep in s, starting at the end of s, and return the part before it, the separator itself, and the part after it. If the separator is not found, return two empty strings and s.	
`s.rsplit([sep [,maxsplit]])`	→ list of strings. Return a list of the words in s, using sep as the delimiter string, starting at the end of the string and working to the front. If maxsplit is given, at most maxsplit splits are done. If sep is not specified, any whitespace string is a separator.	
`s.rstrip([chars])`	Return a copy of the string s with trailing whitespace removed. If chars is given and not None, remove characters in chars instead.	
`s.split([sep [,maxsplit]])`	→ list of strings. Return a list of the words in s, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.	
`s.splitlines([keepends])`	→ list of strings. Return a list of the lines in s, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true.	
`s.startswith(prefix [,start [,end]])`	Return True if s starts with the specified prefix, False otherwise. With optional start, test s beginning at that position. With optional end, stop comparing s at that position. prefix can also be a tuple of strings to try.	
`s.strip([chars])`	Return a copy of the string s with leading and trailing whitespace removed.	
`s.swapcase()`	Return a copy of s with uppercase characters converted to lowercase and vice versa.	
`s.title()`	Return a titlecased version of s	
`s.translate(table [,deletechars])`	Return a copy of the string s in which each character has been mapped through the given translation table. . . .	
`s.upper()`	Return a copy of s converted to uppercase	
`s.zfill(width)`	→ str. Pad a numeric string s with zeros on the left, to fill a field of the specified width. The string s is never truncated.	

- String concatenation: `'abc' 'def' 'ghi'`

TUPLE METHODS

`T.count  T.index`

Method	*Description*	*Notes*
`T.count(value)`	return number of occurrences of value	
`T.index(value, [start, [stop]])`	Return first index of value	

Also see section *Operations on all sequences*

Examples:

```
>>> T = (3, 5, 1, 20, 4)
>>> len(T)
5
>>> T[1:-1]
(5, 1, 20)
>>> a, *b, c = T
>>> a, b, c
(3, [5, 1, 20], 4)
```

LIST METHODS

```
L.append*   L.copy     L.extend*   L.insert*   L.remove*    L.sort*
L.clear*    L.count    L.index     L.pop*      L.reverse*
```

Operations marked with * are in-place.

Method	*Description*	*Notes*
`L.append(obj)`	→ None. Append obj to end.	
`L.clear()`	→ None. Remove all items from L.	
`L.copy()`	Return a shallow copy of L.	
`L.count(value)`	Return number of occurrences of value.	
`L.extend(iterable)`	→ None. Extend list by appending elements from the iterable.	
`L.index(value, [start, [stop]])`	Return first index of value.	
`L.insert(index, object)`	→ None. Insert object before index.	
`L.pop([index])`	Remove and return item at index (default last).	
`L.remove(value)`	→ None. Remove first occurrence of value.	
`L.reverse()`	→ None. Reverse in-place.	
`L.sort(key=None, reverse=False)`	→ None. Sort in-place.	

- List Comprehension

```
L = [e**2 for e in range(10)]

L[:] = [e**2 for e in L]       # change L in place
```

Examples:

```
>>> L = [23, 2, 9, 7, 13]
>>> M = L                 # two references to same object
>>> L[1] = 99
>>> M[1]
99                        # !!!
>>> K = L[:]              # two references to different objects
>>> K[1] = 5
>>> L[1]
99
```

DICTIONARY METHODS

```
D.clear*      D.fromkeys    D.items       D.pop         D.setdefault  D.values
D.copy        D.get         D.keys        D.popitem     D.update*
```

Method	*Description*	*Notes*
`D.clear()`	Remove all items from D; → None	
`D.copy()`	Return a shallow copy of D	
`D.fromkeys(iterable [,value])`	Return a new dictionary with keys from a supplied iterable and values all set to specified value (default None).	
`D.get(k [,d])`	Return D[k] if k in D, else d; d defaults to None.	
`D.items()`	Return a set-like object providing a view on D's items	
`D.keys()`	Return a set-like object providing a view on D's keys	
`D.pop(k [,d])`	Remove given key k, Return corresponding value. If key not found → d if given; else KeyError.	
`D.popitem()`	Remove and return some (key, value) pair	
`D.setdefault(k [,d])`	D.get(k,d), also set D[k]=d if k not in D	
`D.update([E, ]**F)`	Update D from dict/iterable E and F; → None	
`D.values()`	Return an object providing a view on D's values	

Examples

```
>>> D = {0: "Null", 1: "Eins"}
>>> v = D.keys()   # v is a "view"
>>> v
dict_keys([0, 1])
>>> for k in v:
>>>     print(k)
>>> D.values()
dict_values(['Null', 'Eins'])
>>> it = D.items()
>>> it
dict_items([(0, 'Null'), (1, 'Eins')])
>>> for k, v in it:
>>>    print(k, v)
>>> E = { "Hans" : 1234, "Maria" : 4321 }
>>> D.update(E, Thorsten=7890)
```

- D.keys(), D.values(), D.items() return *views*.
- Sort a view with `sorted()`.
- Dictionary comprehension

SET METHODS

```
s.add                          s.intersection (&)             s.remove
s.clear                        s.intersection_update (s &= t) s.symmetric_difference (^)
s.copy                         s.isdisjoint                   s.symmetric_difference_update (^=)
s.difference (-)               s.issubset (<=)                s.union (|)
s.difference_update (-=)       s.issuperset (>=)              s.update (|=)
s.discard                      s.pop
```

Method	*Description*
`s.add(x)`	Add an element to S; do nothing if element exists in S
`s.clear()`	Remove all elements from S
`s.copy()`	→ a new set with a shallow copy of s.
`s.difference(t, ...)`	→ new set with elements in the set that are not in the others.
`s.difference_update(t, ...)`	Update the set, removing elements found in others (`s -= t`)
`s.discard(elem)`	Remove elem in set s if present.
`s.intersection(t, ...)`	→ a new set with elements common to the set and all others (`s & t`)
`s.intersection_udate(t)`	→ set s keeping only elements also found in t (`s &= t`)
`s.disjoint(t)`	→ True if the set has no elements in common with t. Sets are disjoint if and only if their intersection is the empty set.
`s.issubset(t)`	Test whether every element in the set is in t (<=).
`s.issuperset(t)`	Test whether every element in t is in the set (>=)
`s.pop()`	Remove and return an arbitrary element from the set. Raises KeyError if the set is empty.
`s.remove(e)`	Remove element e from the set. Raises KeyError if elem is not contained in the set.
`s.symmetric_difference(t)`	→ a new set with elements in either the set or other but not both (`s ^ t`)
`s.symmetric_difference_update(t)`	Update the set, keeping only elements found in either set, but not in both (`s ^= t`)
`s.union(t, ...)`	Return a new set with elements from the set and all others (|).
`s.update(t, ...)`	Update the set, adding elements from all others (|=)

Example:

```
>>> T = {"e", "a", "g"}        # {} is a dict. Use set() for empty set.
>>> S = set(["a", "b", "c"])
>>> S - T
{'b', 'c'}
>>> S | T
{'a', 'b', 'c', 'e', 'g'}
```

- `a < b` → `a <= b and a != b`
- Set comprehensions
- `frozenset()`

FUNCTIONS

- ```
  def f1(a, b, c):
      return [a, b, c]

  f1(2, 5, 9)
  ```

- ```
  def f2(a, b=5, c=9):
      return [a, b, c]

  f1(2)
  f1(2, 5, 9)
  f1(2, c=3)
  f1(2, b=7)
  f1(2, c=3, b=4)
  ```

- ```
  def f3(a, b, c=5, d=9):
      return (a, b), {'c':c, 'd':d}
  ```

  ```
  def f3(*posargs, **kwargs):
      return posargs, kwargs
  ```
  ```

```
T = (3, 7)
D = { 'c': 14, 'd': 23 }
f3(*T, **D}
```

- ```
  def f4():
      return 'a', 'b', 'c', 'd', 'e'

  m, *n, o = f4()
  ```

BUILTIN FUNCTIONS

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	
  ```

Method	*Description*
`abs(x)`	Returns the absolute value of the number x
`all(iter)`	Return True if all elements of the iterable are true (or if the iterable is empty)
`any(iter)`	Return True if any element of the iterable is true.
`ascii(obj)`	Return a repr string with non-ASCII escaped
`bin(x)`	Convert integer number to binary string
`bool([x])`	→ True when the argument x is true, False otherwise
`bytearray([arg [,encoding [,errors]]])`	Construct an mutable bytearray object from iterable-of-ints/str/bytes-of-buffer/int
`bytes([arg [,encoding [,errors]]])`	Construct an immutable array of bytes
`callable(obj)`	→ whether obj is callable (bool)
`chr(i)`	→ a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.
`classmethod(fn)`	→ method; Convert a function to be a class method.
`compile(source, filename, mode)`	Compile the source (a Python module, statement or expression) into a code object that can be executed by exec() or eval().
`complex(real [,imag])`	
`delattr(obj, name)`	Delete a named attribute on an object; delattr(x, 'y') is equivalent to "del x.y".
`dict([arg])`	
`dir([object])`	If called without an argument, return the names in the current scope. Else, return an alphabetized list of names comprising (some of) the attributes of the given object, and of attributes reachable from it. If the object supplies a method named `__dir__`, it will be used; otherwise the default `dir()` logic is used and returns: For a module object: the module's attributes. For a class object: its attributes, and recursively the attributes of its bases. For any other object: its attributes, its class's attributes, and recursively the attributes of its class's base classes.
`divmod(x, y)`	→ (div, mod); this is the tuple `((x-x%y)/y, x%y)`
`enumerate(iter, start=0)`	→ enumerate object which yields pairs containing a count and a value yielded by the iterable argument: `(0, seq[0]), (1, seq[1]), (2, seq[2]), ...`
`eval(source [,globals [,locals]])`	Evaluate the source in the context of globals and locals. The source may be a string representing a Python expression or a code object as returned by compile().
`exec(object[, globals[, locals]])`	Read and execute code from an object, which can be a string or a code object. The globals and locals are dictionaries, defaulting to the current globals and locals. If only globals is given, locals defaults to it.
`filter(fn, iter)`	→ an iterator yielding those items of iterable for which function(item) is true. If function is None, return the items that are true.
`float([x])`	Convert a string or number to a floating point number, if possible.
`format(value [,format_spec])`	→ formatted string
`frozenset([iter])`	Build an immutable unordered collection of unique elements.
`getattr(obj, name [,default])`	Get a named attribute from an object; getattr(x, 'y') is equivalent to x.y. When a default argument is given, it is returned when the attribute doesn't exist; without it, an exception is raised in that case.
`globals()`	→ the dictionary containing the current scope's global variables.
`hasattr(obj, name)`	→ whether the object has an attribute with the given name.
`hash(obj)`	→ an (int) hash value for the object.
`help([obj])`	This is a wrapper around pydoc.help that provides a helpful message when 'help' is typed at the Python interactive prompt.

`hex(x)`	
`id(obj)`	→ identity of an object (int). This is guaranteed to be unique among simultaneously existing objects.
`input([prompt])`	Read a string from standard input. The trailing newline is stripped. If the user hits EOF (Unix: Ctl-D, Windows: Ctl-Z+Return), raise EOFError. On Unix, GNU readline is used if enabled. The prompt string, if given, is printed without a trailing newline before reading.
`int([number | string [,base])`	Convert a number or string to an integer, or return 0 if no arguments are given. If x is a number, return `x.__int__()`. For floating point numbers, this truncates towards zero.
`isinstance(obj, class-or-type-or-tuple)`	Return whether an object is an instance of a class or of a subclass thereof.
`issubclass(C, B)`	Return whether class C is a subclass (i.e., a derived class) of class B.
`iter(iterable)`	Get an iterator from an object. The argument must supply its own iterator, or be a sequence.
`iter(callable, sentinel)`	Get an iterator from a callable. The callable is called until it returns the sentinel.
`len(obj)`	→ number of items of a sequence or collection.
`list([iter])`	→ new list initialized from iterable's items
`locals()`	Update and return a dictionary containing the current scope's local variables.
`map(func, *iterables)`	→ map object; Make an iterator that computes the function using arguments from each of the iterables. Stops when the shortest iterable is exhausted.
`max(iterable, *[, default=obj, key=func])`	With a single iterable argument, return its biggest item. The default keyword-only argument specifies an object to return if the provided iterable is empty.
`max(arg1, arg2, *args, *[, key=func])`	With two or more arguments, return the largest argument.
`memoryview(obj)`	Create a new memoryview object which references the given object.
`min(iterable, *[, default=obj, key=func])`	→ smallest argument
`next(iter [,default])`	→ next item from the iterator. If default is given and the iterator is exhausted, it is returned instead of raising StopIteration.
`object()`	→ new featureless object. object is a base for all classes.
`oct(number)`	→ octal representation of an integer.
`open(file, **kwargs)`	Open file and return a stream (file object). Raise IOError upon failure. `open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)`
`ord(c)`	→ octal representation of an integer
`pow(x, y [,z])`	With two arguments, equivalent to `x**y`. With three arguments, equivalent to `(x**y) % z`, but may be more efficient (e.g. for ints).
`print(obj, ..., sep, end, file, flush)`	Prints the values to a stream, or to sys.stdout by default.
`property(**kwargs)`	Define managed attributes. See also decorator `@property`
`range(stop)`	→ sequence of numbers from 0 to stop - 1
`range(start, stop [,step])`	→ sequence of numbers from start to stop - 1 by step.
`repr(obj)`	→ canonical string representation of the object. For most objects `eval(repr(object)) == object`
`reversed(sequence)`	→ reverse iterator
`round(x [,ndigits])`	Round a number to a given precision in decimal digits (default 0 digits). This returns an int when called with one argument, otherwise the same type as the number. ndigits may be negative.

`set(iter)`	Build an unordered collection of unique elements.
`setattr(obj, name, val)`	Set a named attribute on an object; setattr(x, 'y', v) is equivalent to x.y = v.
`slice(start, stop [,step])`	Create a slice object. This is used for extended slicing (e.g. a[0:10:2]).
`sorted(iterable, key=None, reverse=False)`	→ new sorted list
`staticmethod(function)`	Convert a function to be a static method
`str([bytes-or-buffer ,[enc [,err]]])`	Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object).
`sum(iter [,start])`	→ sum of an iterable of numbers (NOT strings) plus the value of parameter 'start' (which defaults to 0). When the iterable is empty, return start.
`super([type [,obj-or-type]])`	
`tuple([iter])`	→ tuple initialized from iterable's items
`type(obj)`	→ the obj's type
`vars([obj])`	→ dictionary; Without arguments, equivalent to locals(). With an argument, equivalent to `object.__dict__`.
`zip(*iters)`	→ a zip object whose `.__next__()` method returns a tuple where the i-th element comes from the i-th iterable argument. The `.__next__()` method continues until the shortest iterable in the argument sequence is exhausted and then it raises StopIteration.

Lit.:

- IPython magic: *obj*?, *obj*??
- `https://docs.python.org/3/library/functions.html`

CONTROL STRUCTURES

```
if ...:                      for ...:
   <block>                      <block>
elif ...:                    else:
   <block>                      <block>
else:
   <block>


                             break
while ...:
   <block>                   continue
else:
   <block>                   return

                             yield
```

FILES

```
fo.buffer()       fo.errors          fo.mode           fo.readline()     fo.truncate()
fo.close()        fo.fileno()        fo.name           fo.readlines()    fo.writable()
fo.closed         fo.flush()         fo.newlines       fo.seek()         fo.write()
fo.detach()       fo.isatty()        fo.read()         fo.seekable()     fo.writelines()
fo.encoding       fo.line_buffering  fo.readable()     fo.tell()
```

Method	*Description*
`fo.close()`	Close file
`fo.closed`	True if file closed
`fo.encoding`	Name of the encoding
`fo.errors`	The error setting of the decoder or encoder
`fo.read(size=-1)`	Read up to size bytes from fo. If size is -1 read all bytes.
`fo.readline(size=-1)`	Read until newline or EOF and return a single str. If the stream is already at EOF, an empty string is returned. If size is specified, at most size characters will be read.
`fo.readlines(hint=-1)`	Read and return a list of lines from the stream. hint can be specified to control the number of lines read: no more lines will be read if the total size (in bytes/characters) of all lines so far exceeds hint.
`fo.seek(offset [, whence])`	Change the stream position to the given byte offset. offset is interpreted relative to the position indicated by whence. The default value for whence is SEEK_SET. Values for whence are: SEEK_SET (0), SEEK_CUR (1), SEEK_END (2).
`fo.tell()`	→ current stream position
`fo.write(b)`	Write the given bytes or bytearray object, b, to the underlying raw stream and return the number of bytes written. This can be less than len(b), depending on specifics of the underlying raw stream, and especially if it is in non-blocking mode.
`fo.writelines(lines)`	Write a list of lines to the stream. Line separators are not added, so it is usual for each of the lines provided to have a line separator at the end.

- Open a file with built-in function `open()` return a file object (see BUILTIN FUNCTIONS below).

```
open(file, mode='r', buffering=-1, encoding=None,
     errors=None, newline=None, closefd=True, opener=None)
```

File open *mode*:

```
"r"   Read; position to begin of file.
"r+"  Read and write; position to begin of file.
"w"   Write file; create file if it does not exist. If it exists
      it is truncated. Position to begin of file.
"w+"  Read and write.  Behavior same as 'w' (file is truncated)
"a"   Append to file. Create file if it does not exist.
"a+"  Read and write (append). Create file if it does not exist.
```

Append **b** to mode for **binary files**, e.g. `"rb"`, `"rb+"`, `"wb+"`.

- Print to file object: `print(s, end="", file=fobject)`

Examples

```
(1)   fo = open('file.txt', 'r'):
      for line in fo:
          ...
      fo.close()
```

```
(2)   with open('file.txt', 'r') as fo:
          lines = fo.readlines()
               ...
```

CLASSES AND OBJECTS

Standard methods and operator redefinition in classes

Basic customization

Method	*Description*
`obj.__new__(cls[, ...])`	Called to create a new instance of class cls.
`x.__init__(self[, ...])`	Instance initialization ("'constructor"')
`x.__class__`	Type of an object.
`x.__del__(self)`	Instance destruction ("'destructor"')
`x.__repr__(self)`	Called by the repr() built-in function to compute the "official" string representation of an object.
`x.__str__(self)`	Called by str(object) and the built-in functions format() and print() to compute the "informal" or nicely printable string representation of an object.
`x.__bytes__(self)`	Called by bytes() to compute a byte-string representation of an object.
`x.__format__(self, format_spec)`	Called by the format() built-in function, and by extension, evaluation of formatted string literals and the str.format() method, to produce a "formatted" string representation of an object.
`x.__hash__(self)`	$\rightarrow$ int. Called by built-in function hash() and for operations on members of hashed collections.
`x.__bool__(self)`	Called to implement truth value testing and the built-in operation bool(); should return False or True.
`x.__add__(self, other)`	the + operator
`x.__sub__(self, other)`	the - operator
`x.__mul__(self, other)`	the * operator
`x.__truediv__(self, other)`	the / operator
many more	see `https://docs.python.org/3/reference/datamodel.html`

Rich comparison

`x.__le__(self, y)`	x <= y
`x.__ge__(self, y)`	x >= y
`x.__eq__(self, y)`	x == y
`x.__lt__(self, y)`	x < y
`x.__gt__(self, y)`	x > y
`x.__ne__(self, y)`	x != y

Attribute access

`__getattr__(self, name)`	Called when an attribute access has not found the attribute
`__getattribute__(self, name)`	Called unconditionally to implement attribute accesses for instances of the class.
`__setattr__(self, name, value)`	Called when an attribute assignment is attempted.
`__delattr__(self, name)`	Like `__setattr__()` but for attribute deletion instead of assignment.
`__dir__(self)`	Called when `dir()` is called on the object.
Implementing Descriptors	
`__get__(self, instance, owner)`	Get attribute
`__set__(self, instance, value)`	Set attribute
`__delete__(self, instance)`	Delete attribute

Lit.: https://docs.python.org/3/reference/datamodel.html

CHANGES FROM PYTHON2 TO PYTHON3

- Print is now a function: `print(...)`.
- Division now works as expected.
- "'Backticks"' `` `...` `` removed, use `repr()`.
- All strings now unicode.
- Strings have a `format()` method.
- Use `2to3` tool to convert from Py 2 to Py 3. But: sometimes manual changes necessary.

- When forced to use `Python 2`: Use `-3` option: warn about Python 3.x incompatibilities that 2to3 cannot trivially fix.
- Reload modules with `importlib.reload(<module>)`.
- Many more ...

Lit.: The Conservative Python 3 Porting Guide `http://portingguide.readthedocs.io/en/latest/`

EXCEPTIONS

Exception hierarchy

```
BaseException
   Exception
      AssertionError
      AttributeError
      FloatingPointError
      GeneratorExit
      ImportError
      KeyError
      IndexError
      NameError
      OSError (see attributes errno and strerror)
      OverflowError
      StopIteration
      SyntaxError
      TypeError
      ValueError
      ZeroDivisionError
      Warning
         DeprecationWarning
         ...
      many more...
      user define exceptions...
```

Lit.: https://docs.python.org/3/library/exceptions.html

Example:

```
print("*** First exception")

try:
    1/0
except ZeroDivisionError as e:
    print e      # *** First exception
                 # integer division or modulo by zero


print("*** Second exception")

class MyExc(Exception):
    def __str__(self):
        return "Instance of MyExc"

try:
    raise MyExc()
except MyExc as x:
    print x      # *** Second exception
                 # Instance of MyExc
```

MISCELLANEOUS

- Assert

```
>>> pressure = 20.0
>>> assert (pressure <= 10.0), "Alert. Too much pressure!"
AssertionError: Alert. Too much pressure!
```

PACKAGE MANAGEMENT AND INSTALLATION

- distutils – basic support to create and install distributions (introduced `setup.py`)

```
python setup.py --help-commands

build         build_ext       build_py        clean          install
sdist         upload check    saveopts        bdist          bdist_wheel
build_sphinx  test            bdist_wininst   [py2exe]       ...
```

- setuptools – enhancements to distutils
  - Create wheels with `bdist_wheels` setuptools extension
  - Do not use `easy_install` and `eggs`!
- wheel – Binary archive format (successor of `eggs`)
  - `wheel` utility installs and verifies wheels
  - newer versions of `pip` also work with wheels
- pip (pip3)
  - Installs source distributions (sdist/.tar.gz) and binary distributions (wheels/.whl)
  - Help

```
pip --help, pip install --help

pip list             list all installed packages
pip search <str>     Search PyPI for <str>
pip show <pkg>       show pkg info
pip install <pkg>    install package
pip install -U <pkg> upgrade package
pip uninstall <pkg>  remove packge
...
```

  - Trick to choose Python version:

```
python3.6 -m pip install ...   # Linux
py -3.6 -m pip install ...     # Windows
```

- Virtual Environments
  - Python 3.6 and higher

```
python3 -m venv new_env       # Windows py -3.6 -m venv new_env
source new_env/bin/activate   # Windows new_env\Scripts\activate.bat
...
deactivate
```

The `pyvenv` tool introduced in Py 3.x is deprecated in 3.6.

– Python 2: virtualenv

```
virtualenv <DIR>
source <DIR>/bin/activate
```

- conda – comes with Anaconda (https://www.continuum.io). Combines functionality of `pip` and `venv`.
- Use `twine` to interact with PyPI.
- `http://www.pyinstaller.org` freezes programs into standalone executables.

Lit.: [9]

DOCUMENT CREATION

- Restructured Text (rst2html, rst2latex, ...)
  http://docutils.sourceforge.net
- Sphinx
  http://www.sphinx-doc.org

STANDARDLIBRARY OVERVIEW

"batteries included" | `https://docs.python.org/3/library`

“Python 3 module of the week”: `https://pymotw.com/3`

Following list is from https://docs.python.org/3/py-modindex.html

```
__future__    Future statement definitions
__main__      The environment where the top-level script is run.
_dummy_thread Drop-in replacement for the _thread module.
_thread       Low-level threading API.

abc       Abstract base classes according to PEP 3119.
aifc      Read and write audio files in AIFF or AIFC format.
argparse  Command-line option and argument parsing library.
array     Space efficient arrays of uniformly typed numeric values.
ast       Abstract Syntax Tree classes and manipulation.
asynchat  Support for asynchronous command/response protocols.
asyncio   Asynchronous I/O, event loop, coroutines and tasks.
asyncore  A base class for developing asynchronous socket handling services.
atexit    Register and execute cleanup functions.
audioop   Manipulate raw audio data.

base64    RFC 3548: Base16, Base32, Base64 Data Encodings; Base85 and Ascii85
bdb       Debugger framework.
binascii  Tools for converting between binary and various ASCII-encoded binary
          representations.
binhex    Encode and decode files in binhex4 format.
bisect    Array bisection algorithms for binary searching.
builtins  The module that provides the built-in namespace.
bz2       Interfaces for bzip2 compression and decompression.

calendar  Functions for working with calendars, including some emulation of the Unix cal program.
cgi       Helpers for running Python scripts via the Common Gateway Interface.
```

```
cgitb     Configurable traceback handler for CGI scripts.
chunk     Module to read IFF chunks.
cmath     Mathematical functions for complex numbers.
cmd       Build line-oriented command interpreters.
code      Facilities to implement read-eval-print loops.
codecs    Encode and decode data and streams.
codeop    Compile (possibly incomplete) Python code.
collections Container datatypes
colorsys    Conversion functions between RGB and other color systems.
compileall  Tools for byte-compiling all Python source files in a directory tree.
concurrent
concurrent.futures Launching parallel tasks.
configparser       Configuration file parser.
contextlib  Utilities for with-statement contexts.
copy        Shallow and deep copy operations.
copyreg     Register pickle support functions.
cProfile
crypt (Unix)  The crypt() function used to check Unix passwords.
csv           Write and read tabular data to and from delimited files.
ctypes        A foreign function library for Python.
curses (Unix) An interface to the curses library, providing portable terminal handling.

datetime      Basic date and time types.
dbm           Interfaces to various Unix "database" formats.
decimal       Implementation of the General Decimal Arithmetic Specification.
difflib       Helpers for computing differences between objects.
dis           Disassembler for Python bytecode.
distutils     Support for building and installing Python modules into an existing Python
              installation.
doctest       Test pieces of code within docstrings.
dummy_threading   Drop-in replacement for the threading module.

email      Package supporting the parsing, manipulating, and generating email messages.
encodings  Package for standard Python encodings (e.g. ascii, iso, utf, ...)
ensurepip  Bootstrapping the "pip" installer into an existing Python installation or
           virtual environment.
enum       Implementation of an enumeration class.
errno      Standard errno system symbols.

faulthandler  Dump the Python traceback.
fcntl (Unix)  The fcntl() and ioctl() system calls.
filecmp       Compare files efficiently.
fileinput     Loop over standard input or a list of files.
fnmatch       Unix shell style filename pattern matching.
fpectl (Unix) Provide control for floating point exception handling.
fractions     Rational numbers.
ftplib        FTP protocol client (requires sockets).
functools     Higher-order functions and operations on callable objects.

gc       Interface to the cycle-detecting garbage collector.
getopt   Portable parser for command line options; support both short and long option names.
getpass  Portable reading of passwords and retrieval of the userid.
gettext  Multilingual internationalization services.
glob     Unix shell style pathname pattern expansion.
grp (Unix)  The group database (getgrnam() and friends).
gzip        Interfaces for gzip compression and decompression using file objects.
```

```
hashlib  Secure hash and message digest algorithms.
heapq    Heap queue algorithm (a.k.a. priority queue).
hmac     Keyed-Hashing for Message Authentication (HMAC) implementation
html     Helpers for manipulating HTML.
http     HTTP status codes and messages

imaplib    IMAP4 protocol client (requires sockets).
imghdr     Determine the type of image contained in a file or byte stream.
importlib  The implementation of the import machinery.
inspect    Extract information and source code from live objects.
io         Core tools for working with streams.
ipaddress  IPv4/IPv6 manipulation library.
itertools  Functions creating iterators for efficient looping.

json       Encode and decode the JSON format.

keyword    Test whether a string is a keyword in Python.

lib2to3    the 2to3 library
linecache  This module provides random access to individual lines from text files.
locale     Internationalization services.
logging    Flexible event logging system for applications.
lzma       A Python wrapper for the liblzma compression library.

macpath    Mac OS 9 path manipulation functions.
mailbox    Manipulate mailboxes in various formats
mailcap    Mailcap file handling.
marshal    Convert Python objects to streams of bytes and back (with different constraints).
math       Mathematical functions (sin() etc.).
mimetypes  Mapping of filename extensions to MIME types.
mmap       Interface to memory-mapped files for Unix and Windows.
modulefinder  Find modules used by a script.
msilib (Windows)   Creation of Microsoft Installer files, and CAB files.
msvcrt (Windows)   Miscellaneous useful routines from the MS VC++ runtime.
multiprocessing    Process-based parallelism.

netrc        Loading of .netrc files.
nis (Unix)   Interface to Sun's NIS (Yellow Pages) library.
nntplib      NNTP protocol client (requires sockets).
numbers      Numeric abstract base classes (Complex, Real, Integral, etc.).

operator     Functions corresponding to the standard operators.
os           Miscellaneous operating system interfaces.
ossaudiodev (Linux, FreeBSD)   Access to OSS-compatible audio devices.

parser        Access parse trees for Python source code.
pathlib       Object-oriented filesystem paths
pdb           The Python debugger for interactive interpreters.
pickle        Convert Python objects to streams of bytes and back.
pickletools   Contains extensive comments about the pickle protocols and pickle-machine
              opcodes, as well as some useful functions.
pipes (Unix)  A Python interface to Unix shell pipelines.
pkgutil       Utilities for the import system.
platform      Retrieves as much platform identifying data as possible.
plistlib      Generate and parse Mac OS X plist files.
poplib        POP3 protocol client (requires sockets).
posix (Unix)  The most common POSIX system calls (normally used via module os).
```

```
pprint        Data pretty printer.
profile       Python source profiler.
pstats        Statistics object for use with the profiler.
pty (Linux)   Pseudo-Terminal Handling for Linux.
pwd (Unix)    The password database (getpwnam() and friends).
py_compile    Generate byte-code files from Python source files.
pyclbr        Supports information extraction for a Python class browser.
pydoc         Documentation generator and online help system.

queue         A synchronized queue class.
quopri        Encode and decode files using the MIME quoted-printable encoding.

random   Generate pseudo-random numbers with various common distributions.
re       Regular expression operations.
readline (Unix)   GNU readline support for Python.
reprlib           Alternate repr() implementation with size limits.
resource (Unix)   An interface to provide resource usage information on the current process.
rlcompleter       Python identifier completion, suitable for the GNU readline library.
runpy             Locate and run Python modules without importing them first.

sched    General purpose event scheduler.
secrets  Generate secure random numbers for managing secrets.
select   Wait for I/O completion on multiple streams.
selectors  High-level I/O multiplexing.
shelve   Python object persistence.
shlex    Simple lexical analysis for Unix shell-like languages.
shutil   High-level file operations, including copying.
signal   Set handlers for asynchronous events.
site     Module responsible for site-specific configuration.
smtpd    A SMTP server implementation in Python.
smtplib  SMTP protocol client (requires sockets).
sndhdr   Determine type of a sound file.
socket   Low-level networking interface.
socketserver   A framework for network servers.
spwd (Unix)    The shadow password database (getspnam() and friends).
sqlite3  A DB-API 2.0 implementation using SQLite 3.x.
ssl      TLS/SSL wrapper for socket objects
stat     Utilities for interpreting the results of os.stat(), os.lstat() and os.fstat().
statistics  Mathematical statistics functions
string      Common string operations.
stringprep  String preparation, as per RFC 3453
struct      Interpret bytes as packed binary data.
subprocess  Subprocess management.
sunau       Provide an interface to the Sun AU sound format.
symbol      Constants representing internal nodes of the parse tree.
symtable    Interface to the compiler's internal symbol tables.
sys         Access system-specific parameters and functions.
sysconfig   Python's configuration information
syslog (Unix)   An interface to the Unix syslog library routines.

tabnanny   Tool for detecting white space related problems in Python source files
           in a directory tree.
tarfile    Read and write tar-format archive files.
telnetlib  Telnet client class.
tempfile   Generate temporary files and directories.
termios (Unix)  POSIX style tty control.
test       Regression tests package containing the testing suite for Python.
```

```
textwrap   Text wrapping and filling
threading  Thread-based parallelism.
time       Time access and conversions.
timeit     Measure the execution time of small code snippets.
tkinter    Interface to Tcl/Tk for graphical user interfaces
token      Constants representing terminal nodes of the parse tree.
tokenize   Lexical scanner for Python source code.
trace      Trace or track Python statement execution.
traceback  Print or retrieve a stack traceback.
tracemalloc   Trace memory allocations.
tty (Unix)    Utility functions that perform common terminal control operations.
turtle        An educational framework for simple graphics applications
turtledemo    A viewer for example turtle scripts
types         Names for built-in types.
typing        Support for type hints (see PEP 484).

unicodedata   Access the Unicode Database.
unittest      Unit testing framework for Python.
urllib
urllib.request     for opening and reading URLs
urllib.error       containing the exceptions raised by urllib.request
urllib.parse       for parsing URLs
urllib.robotparser for parsing robots.txt files

uu            Encode and decode files in uuencode format.
uuid          UUID objects (universally unique identifiers) according to RFC 4122

venv    Creation of virtual environments.

warnings    Issue warning messages and control their disposition.
wave        Provide an interface to the WAV sound format.
weakref     Support for weak references and weak dictionaries.
webbrowser  Easy-to-use controller for Web browsers.
winreg (Windows)    Routines and objects for manipulating the Windows registry.
winsound (Windows)  Access to the sound-playing machinery for Windows.
wsgiref             WSGI Utilities and Reference Implementation.

xdrlib    Encoders and decoders for the External Data Representation (XDR).
xml       Package containing XML processing modules
xmlrpc

zipapp     Manage executable python zip archives
zipfile    Read and write ZIP-format archive files.
zipimport  support for importing Python modules from ZIP archives.
zlib       Low-level interface to compression and decompression routines compatible
           with gzip.
```

IMPORTANT MODULES

Module datetime

```
import datetime
today = datetime.date.today()
chrismas = datetime.date(2016, 12, 24)
print(chrismas - today)    # 299 days, 0:00:00
dT = datetime.timedelta(days=8)
print(chrismas + dT)   # 2017-01-01
```

Lit.: https://docs.python.org/3/library/datetime.html

Module operators

```
operator.__abs__        operator.__getitem__    operator.__ipow__       operator.__neg__
operator.__add__        operator.__gt__         operator.__irshift__    operator.__not__
operator.__all__        operator.__iadd__       operator.__isub__       operator.__or__
operator.__and__        operator.__iand__       operator.__itruediv__   operator.__package__
operator.__builtins__   operator.__iconcat__    operator.__ixor__       operator.__pos__
operator.__cached__     operator.__ifloordiv__  operator.__le__         operator.__pow__
operator.__concat__     operator.__ilshift__    operator.__loader__     operator.__rshift__
operator.__contains__   operator.__imatmul__    operator.__lshift__     operator.__setitem__
operator.__delitem__    operator.__imod__       operator.__lt__         operator.__spec__
operator.__doc__        operator.__imul__       operator.__matmul__     operator.__sub__
operator.__eq__         operator.__index__      operator.__mod__        operator.__truediv__
operator.__file__       operator.__inv__        operator.__mul__        operator.__xor__
operator.__floordiv__   operator.__invert__     operator.__name__
operator.__ge__         operator.__ior__        operator.__ne__
```

Module pickle

```
import pickle
...
fo = open("picklefile", "wb")
pickle.dump(data, fo)
fo.close()
...
fo = open("picklefile", "rb")
data = pickle.load(fo)
fo.close()
...
```

Module random

```
import random
```

Function	*Description*
`seed([x])`	Initialize the basic random number generator.
`randrange([start], stop[, step])`	Return a randomly selected element from range(start, stop, step).
`randint(a, b)`	Return a random integer N such that `a <= N <= b`.
`choice(seq)`	Return a random element from the non-empty sequence seq. If seq is empty, raises IndexError.
`random()`	→ next random floating point number in the range `[0.0, 1.0)`.
`sample(population, k)`	Chooses k unique random elements from a population sequence or set.
`shuffle(x[, random])`	Shuffle the sequence x in place.

Example:

```
import random
R = [random.randint(1, 6) for i in range(100)]
```

Module re

```
import re
```

Function	*Description*
`m = re.match(pattern, string, flags=0)`	Try to apply the pattern at the start of the string, returning a match object, or None if no match was found.
`re.findall(pattern, string, flags=0)`	Return a list of all non-overlapping matches in the string.
`m.group([group1, ...])`	→ str or tuple. Return subgroup(s) of the match by indices or names. For 0 returns the entire match.
`m.groups([default=None])`	→ tuple. Return a tuple containing all the subgroups of the match, from 1. The default argument is used for groups that did not participate in the match.

```
.     any character               +    1 or more repetitions
\d    decimal digit               *    0 or more repetitions
\D    opposite of \d              ^    start of string
\s    whitespace character        $    end of string
\S    opposite of \s
\w    Word (incl. 0-9 and _)
\W    opposite of \w              []   set of chars
\Z    end of string               a|b  match a or b
```

Example:

```
>>> m = re.match(r"(\d+)\.(\d+)", "24.1632")
>>> m.groups()
('24', '1632')
>>> m.group(0)   # whole match
'24.1632'
>>> m.group(1)
'24'
>>> m.group(2)
'1632'
>>> re.findall("(Bi\w*)+", "Boston Bilbao Chicago Muenchen Birmingham")
['Bilbao', 'Birmingham']
```

Lit.:

- `https://docs.python.org/3.3/library/re.html`
- `https://docs.python.org/3/howto/regex.html`
- `https://regex101.com`

Module time

```
import time
```

Function	*Description*
`time()`	→ time in seconds since the epoch as a floating point number, e.g. 1456693297.0732343
`asctime()`	→ e.g. 'Sun Feb 28 21:59:18 2016'
`localtime([secs])`	→ a `struct_time` object, e.g. `tm_year=2016, tm_mon=2, tm_mday=28, tm_hour=22, tm_min=3, tm_sec=59, tm_wday=6, tm_yday=59, tm_isdst=0`.
`strftime(format[, t])`	→ Convert a tuple or `struct_time` representing a time as returned by `gmtime()` or `localtime()` to a string as specified by the format argument.

Example:

```
>>> import time
>>> time.strftime("%b %d %Y %H:%M:%S", time.gmtime())
'Feb 12 2018 07:15:42'
```

See also the `timeit` Module.

Lit.: https://docs.python.org/3/library/time.html

Numeric and Scientific Python

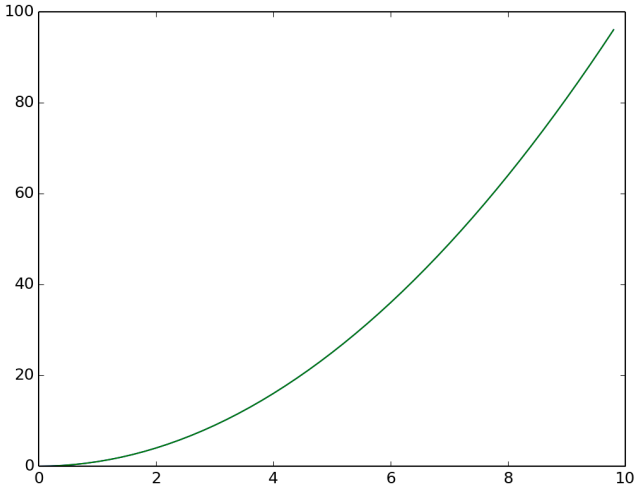
- Numpy

```
>>> import numpy as np
>>> x = np.arange(0.8, 2.2, 0.1)
>>> x
array([ 0.8, 0.9, 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2., 2.1])
>>> z = np.linspace(0.2, 1.4, 4)
>>> z
array([ 0.2,  0.6,  1. ,  1.4])
```

  Lit.: `http://www.numpy.org`

- Matplotlib
  - Depends on NumPy.
  - `from matplotlib import pyplot` (use in scripts)
  - `from matplotlib import mlab` (MATLAB compatible interface)
  - "'pylab"' = Matlab-like interface to NumPy and Matplotlib – mainly for interactive work
    `import matplotlib.pylab` *or*
    `import pylab` *or*
    `from pylab import *` *or*
    `ipython --pylab` *or*
    `%pylab` IPython magic command
  - `help(pylab)`

  Example:




```
from pylab import *
x = arange(0, 10.0, 0.2)
y = x ** 2.0   # x and y are arrays!
plot(x, y)
savefig('plot', dpi=150)   # plot.png
```

  (opt.: `%matplotlib inline`)

  Lit.: `http://matplotlib.org`

- SciPy

```
>>> import scipy
```

Lit.: `http://www.scipy.org`

- Imports for numerical programs (keep namespaces separate!):

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
```

Tool Quickrefs: Conda, Spyder, Jupyter NB

Conda

```
conda search <package>                  pip search <package>

conda install <package>                 pip install <package>
conda install python=x.x
conda install --channel <name> <module>

conda create --name <env> python
source activate <env>
source deactivate                       deactivate
conda list --name <env>                 pip list
conda list --export                     pip freeze
conda info --envs (list all envs)       lsvirtualenv (virtualenv wrapper)
conda install python=x.x
conda update python *
conda install pip                       pip install conda
                                        pip show conda
conda remove <package>
conda remove --name <env> <package>
```

Example: Creates environment `testenv` in `/anaconda3/envs/`:

```
conda create --name testenv python
source activate testenv
(testenv) hhoegl@e11:~/anaconda3/envs$
```

Spyder keyboard shortcuts

```
Quit       C-Q          Go to def   C-G
Restart    A-Sh-R       Find text   C-F
Run        F5           Find next   F3
Fullscreen F11          Code complete  C-Space
Breakpoint F12          Undo        C-Z
Debug      C-F5         Indent      Tab
Block comment  C-4      Unindent    Sh-Tab
Block uncomm   C-5      many more, see Tools -> Preferences menu in Spyder
```

Jupyter notebook keyoard shortcuts

Command mode (press Esc to enable)

```
Enter edit mode    Enter
Run cell           Sh-Enter
Run cell, ins bel  Alt-Enter
```

```
To code            Y
To markdown        M
To raw             R
To heading 1       1
Save + checkpoint  Ctrl-S
Toggle line nmb    L
Toggle output      O
Close pager        ESC
Shortcut help      H
Interrupt kernel   I,I
Restart kernel     0,0
Scroll down        Space
Scroll up          Sh-Space
```

Edit mode (press Enter to enable)

```
Code completion    Tab
Tooltip            Sh-Tab
Indent             Ctrl-]
Dedent             Ctrl-[
Select all         Ctrl-A
Undo               Ctrl-Z
Toggle comment     Ctrl-/
```

References

[1] Non-Programmer's Tutorial for Python 3
https://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_3

[2] Bernd Klein, http://python-kurs.eu

[3] Swaroop C H, A Byte of Python, https://python.swaroopch.com

[4] Mark Pilgrim, Dive Into Python 3
http://www.diveintopython3.net

[5] Unicode HOWTO
https://docs.python.org/3/howto/unicode.html

[6] Python 3 Standard Documentation
https://docs.python.org/3

[7] Michael Weigend, Python ge-packt. 7. Auflage, mitp Verlag, 2018.

[8] David Beazley, Python Essential Reference, Addison-Wesley 2009.

[9] Python Packaging User Guide
https://packaging.python.org

[10] The Hitchhiker's Guide to Python
http://docs.python-guide.org/en/latest

Your notes:

Your notes:

Your notes: