

Python 3 Quick Reference

Hubert Högl

Revision 0.5, 2019-01-24

2018, 2019

<https://github.com/huberthoegl/pqr3>

Contents		Functions	11
Environment Variables	2	Builtin Functions	12
Keywords	2	Control Structures	14
Python	2	Files	15
IPython	2	Classes and Objects	16
Source Encoding	2	Changes from Python2 to Python3	17
Numbers	3	Exceptions	17
Operations on numbers	3	Miscellaneous	18
Integer	3	Document creation	18
Float	4	Standardlibrary Overview	18
Complex	4	Important Modules	23
Assignments	4	Numeric and Scientific Python	25
Comparisons	4	TOOLS	26
Sequences, Mappings and Collections	4	Package Distribution	26
Operations on all sequences	5	IPython	27
Operations on mutable sequences	5	Jupyter	28
Operations on mappings	6	Conda/Pip Package Management	29
Operations on Sets	6	Virtual Environments	29
String Methods	6	Ipdb	30
Tuple Methods	8	Spyder	31
List Methods	9	Windows Keys	31
Dictionary Methods	9	Firefox Keys	31
Set Methods	10	Literature	31

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

This text ist licensed under the Creative Commons License CC-BY-NC-SA (please see <https://creativecommons.org>).

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	

PYTHON

```
hhoegl@e11 ~$ python
Python 3.7.1 (default, Dec 14 2018, 19:28:38)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

- Basic Python prompt.
- Tab Completion.
- Use only for short "throw-away" interactive work.
- For comfortable interactive work use IPython.
- Windows: Launcher py [Version] pgm.py, newest Version default, -2 for Python 2.

IPYTHON

```
hhoegl@e11 ~$ ipython
Python 3.7.1 (default, Dec 14 2018, 19:28:38)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.2.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]: _
```

- Nice tab completion based on prompt_toolkit and ptpython.
- See page ?? for more information.

SOURCE ENCODING

- 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"
```

- Unicode reader: [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, remainder	
$-X$, $+X$	sign	
$X Y$, $X \& Y$	bitwise or, bitwise and	
$X \wedge Y$	exor	
$X \ll n$, $n \gg n$	shift	
$\sim X$	bitwise negate	
$X ** Y$	pow	
<code>abs(X)</code>	absolute value	
<code>int(X)</code>	type conversion	
<code>float(X)</code>	type conversion	
<code>complex(X)</code>	type conversion	
<code>divmod(X, Y)</code>	integer division and remainder	
<code>pow(X, Y, [Z])</code>	identical with <code>**</code>	

- 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
```

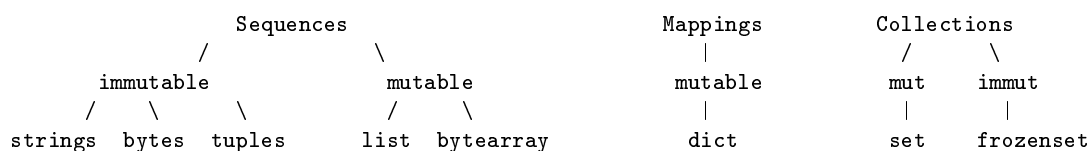
COMPARISONS

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



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
<code>len(S)</code>	Number of elements in S	
<code>x in S</code>	x member in S	
<code>x not in S</code>	x not member in S	
<code>S1 + S2</code>	Concatenation	
<code>S * n</code>	Sequence times int	
<code>n * S</code>	Int times sequence	
<code>S[i]</code>	Index Operator	
<code>S[i:j]</code>	Slice Operator	
<code>S[i:j:k]</code>	Slice Operator	
<code>S.index(x)</code>	Get index of element x	
<code>min(S)</code>	Minimum	
<code>max(S)</code>	Maximum	
<code>iter(S)</code>	Return Iterator	
<code>for x in S:</code>	Iteration	
<code>[e for e in S]</code>	List comprehension	
<code>map(f, S)</code>	Functional programming	
<code>filter(f, S)</code>	Functional programming	

Operations on mutable sequences

Operation	Description	Notes
<code>S[i] = x</code>	Assign x at index i in S	
<code>S[i:j] = T</code>	Assign T to slice	
<code>S[i:j:k] = T</code>	Assign T to slice	
<code>del S[i]</code>	Delete	
<code>del S[i:j]</code>	Delete	
<code>del S[i:j:k]</code>	Delete	

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

Operations on Mappings

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

Operations on Sets

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

See also section SET METHODS below.

STRING METHODS

IPython Hilfe (`s.<tab>`)

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

Method	Description	Notes
<code>s.capitalize()</code>	Returns a copy of s with its first character capitalized, and the rest of the characters lowercase.	
<code>s.casefold()</code>	Return a version of s suitable for caseless comparisons.	
<code>s.center(width[, fillchar])</code>	Returns a copy of s centered in a string of length <i>width</i> , surrounded by the appropriate number of <i>fillchar</i> characters.	

<code>s.count(sub[, start[, end]])</code>	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.	
<code>s.encode([encoding[, errors]])</code>	Encode s using the codec registered for encoding. Return bytes.	
<code>s.endswith(suffix[, start[, end]])</code>	Return True if s ends with the specified suffix, False otherwise.	
<code>s.expandtabs([tabsize])</code>	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.	
<code>s.find(sub[, start[, end]])</code>	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.	
<code>s.format(*args, **kwargs)</code>	see https://pyformat.info	
<code>s.format_map(mapping)</code>	Return a formatted version of s, using substitutions from mapping.	
<code>s.index(sub[, start[, end]])</code>	Like s.find() but raise ValueError when the substring is not found.	
<code>s.isalnum()</code>	Return True if all characters in s are alphanumeric and there is at least one character in s, False otherwise.	
<code>s.isalpha()</code>	Return True if all characters in s are alphabetic and there is at least one character in s, False otherwise.	
<code>s.isdecimal()</code>	Return True if there are only decimal characters in s, False otherwise.	
<code>s.isdigit()</code>	Return True if all characters in s are digits and there is at least one character in s, False otherwise.	
<code>s.isidentifier()</code>	Return True if s is a valid identifier according to the language definition.	
<code>s.islower()</code>	Return True if all cased characters in s are lowercase and there is at least one cased character in s, False otherwise.	
<code>s.isnumeric()</code>	Return True if there are only numeric characters in s, False otherwise.	
<code>s.isprintable()</code>	Return True if all characters in s are considered printable in repr() or s is empty, False otherwise.	
<code>s.isspace()</code>	Return True if all characters in s are whitespace and there is at least one character in s, False otherwise.	
<code>s.istitle()</code>	Return True if s is a titlecased string	
<code>s.isupper()</code>	Return True if all cased characters in s are uppercase and there is at least one cased character in s, False otherwise.	
<code>s.join(iterable)</code>	Return a string which is the concatenation of the strings in the iterable. The separator between elements is s.	
<code>s.ljust(width[, fillchar])</code>	Return s left-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).	
<code>s.lower()</code>	Return a copy of the string s converted to lowercase.	
<code>s.lstrip([chars])</code>	Return a copy of the string s with leading whitespace removed. If chars is given and not None, remove characters in chars instead.	
<code>s.maketrans()</code>	Return a translation table usable for s.translate().	
<code>s.partition(sep)</code>	→ (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.	
<code>s.replace(old, new[, count])</code>	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.	
<code>s.rfind(sub[, start[, len]])</code>	Return highest index in s where substring sub is found, such that sub is contained within s[start:end].	

<code>s.rindex(sub [,start [,len]])</code>	Like <code>s.rfind()</code> but raise <code>ValueError</code> when the substring is not found.	
<code>s.rjust(width [,fillchar])</code>	Return <code>s</code> right-justified in a string of length <code>width</code> . Padding is done using the specified fill character (default is a space).	
<code>s.rpartition(sep)</code>	Search for the separator <code>sep</code> in <code>s</code> , starting at the end of <code>s</code> , 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 <code>s</code> .	
<code>s.rsplit([sep [,maxsplit]])</code>	→ list of strings. Return a list of the words in <code>s</code> , using <code>sep</code> as the delimiter string, starting at the end of the string and working to the front. If <code>maxsplit</code> is given, at most <code>maxsplit</code> splits are done. If <code>sep</code> is not specified, any whitespace string is a separator.	
<code>s.rstrip([chars])</code>	Return a copy of the string <code>s</code> with trailing whitespace removed. If <code>chars</code> is given and not <code>None</code> , remove characters in <code>chars</code> instead.	
<code>s.split([sep [,maxsplit]])</code>	→ list of strings. Return a list of the words in <code>s</code> , using <code>sep</code> as the delimiter string. If <code>maxsplit</code> is given, at most <code>maxsplit</code> splits are done. If <code>sep</code> is not specified or is <code>None</code> , any whitespace string is a separator and empty strings are removed from the result.	
<code>s.splitlines([keepends])</code>	→ list of strings. Return a list of the lines in <code>s</code> , breaking at line boundaries. Line breaks are not included in the resulting list unless <code>keepends</code> is given and <code>true</code> .	
<code>s.startswith(prefix [,start [,end]])</code>	Return <code>True</code> if <code>s</code> starts with the specified prefix, <code>False</code> otherwise. With optional <code>start</code> , test <code>s</code> beginning at that position. With optional <code>end</code> , stop comparing <code>s</code> at that position. <code>prefix</code> can also be a tuple of strings to try.	
<code>s.strip([chars])</code>	Return a copy of the string <code>s</code> with leading and trailing whitespace removed.	
<code>s.swapcase()</code>	Return a copy of <code>s</code> with uppercase characters converted to lowercase and vice versa.	
<code>s.title()</code>	Return a titlecased version of <code>s</code>	
<code>s.translate(table [,deletechars])</code>	Return a copy of the string <code>s</code> in which each character has been mapped through the given translation table. ...	
<code>s.upper()</code>	Return a copy of <code>s</code> converted to uppercase	
<code>s.zfill(width)</code>	→ str. Pad a numeric string <code>s</code> with zeros on the left, to fill a field of the specified width. The string <code>s</code> is never truncated.	

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

TUPLE METHODS

`T.count` `T.index`

Method	Description	Notes
<code>T.count(value)</code>	return number of occurrences of value	
<code>T.index(value, [start, [stop]])</code>	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	

<code>D.keys()</code>	Return a set-like object providing a view on D's keys	
<code>D.pop(k [,d])</code>	Remove given key k, Return corresponding value. If key not found → d if given; else <code>KeyError</code> .	
<code>D.popitem()</code>	Remove and return some (key, value) pair	
<code>D.setdefault(k [,d])</code>	<code>D.get(k,d)</code> , also set <code>D[k]=d</code> if k not in D	
<code>D.update([E,]**F)</code>	Update D from dict/iterable E and F; → None	
<code>D.values()</code>	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

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

Method	Description
<code>s.add(x)</code>	Add an element to S; do nothing if element exists in S
<code>s.clear()</code>	Remove all elements from S
<code>s.copy()</code>	→ a new set with a shallow copy of s.
<code>s.difference(t, ...)</code>	→ new set with elements in the set that are not in the others.
<code>s.difference_update(t, ...)</code>	Update the set, removing elements found in others (<code>s -= t</code>)
<code>s.discard(elem)</code>	Remove elem in set s if present.
<code>s.intersection(t, ...)</code>	→ a new set with elements common to the set and all others (<code>s & t</code>)
<code>s.intersection_update(t)</code>	→ set s keeping only elements also found in t (<code>s &= t</code>)
<code>s.disjoint(t)</code>	→ True if the set has no elements in common with t. Sets are disjoint if and only if their intersection is the empty set.

<code>s.issubset(t)</code>	Test whether every element in the set is in t (\leq).
<code>s.issuperset(t)</code>	Test whether every element in t is in the set (\geq)
<code>s.pop()</code>	Remove and return an arbitrary element from the set. Raises <code>KeyError</code> if the set is empty.
<code>s.remove(e)</code>	Remove element e from the set. Raises <code>KeyError</code> if elem is not contained in the set.
<code>s.symmetric_difference(t)</code>	\rightarrow a new set with elements in either the set or other but not both ($s \oplus t$)
<code>s.symmetric_difference_update(t)</code>	Update the set, keeping only elements found in either set, but not in both ($s \oplus= t$)
<code>s.union(t, ...)</code>	Return a new set with elements from the set and all others (\cup).
<code>s.update(t, ...)</code>	Update the set, adding elements from all others ($\cup=$)

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'}
```

- $a < b \rightarrow a \leq b$ and $a \neq b$
- 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 <code>__dir__</code> , it will be used; otherwise the default <code>dir()</code> 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.

<code>divmod(x, y)</code>	→ (div, mod); this is the tuple $((x-x\%y)/y, x\%y)$
<code>enumerate(iter, start=0)</code>	→ 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]), ...
<code>eval(source [,globals [,locals]])</code>	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().
<code>exec(object[, globals[, locals]])</code>	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.
<code>filter(fn, iter)</code>	→ an iterator yielding those items of iterable for which function(item) is true. If function is None, return the items that are true.
<code>float([x])</code>	Convert a string or number to a floating point number, if possible.
<code>format(value [,format_spec])</code>	→ formatted string
<code>frozenset([iter])</code>	Build an immutable unordered collection of unique elements.
<code>getattr(obj, name [,default])</code>	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.
<code>globals()</code>	→ the dictionary containing the current scope's global variables.
<code>hasattr(obj, name)</code>	→ whether the object has an attribute with the given name.
<code>hash(obj)</code>	→ an (int) hash value for the object.
<code>help([obj])</code>	This is a wrapper around pydoc.help that provides a helpful message when 'help' is typed at the Python interactive prompt.
<code>hex(x)</code>	
<code>id(obj)</code>	→ identity of an object (int). This is guaranteed to be unique among simultaneously existing objects.
<code>input([prompt])</code>	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.
<code>int([number string [,base]])</code>	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.
<code>isinstance(obj, class-or-type-or-tuple)</code>	Return whether an object is an instance of a class or of a subclass thereof.
<code>issubclass(C, B)</code>	Return whether class C is a subclass (i.e., a derived class) of class B.
<code>iter(iterable)</code>	Get an iterator from an object. The argument must supply its own iterator, or be a sequence.
<code>iter(callable, sentinel)</code>	Get an iterator from a callable. The callable is called until it returns the sentinel.
<code>len(obj)</code>	→ number of items of a sequence or collection.
<code>list([iter])</code>	→ new list initialized from iterable's items
<code>locals()</code>	Update and return a dictionary containing the current scope's local variables.
<code>map(func, *iterables)</code>	→ map object; Make an iterator that computes the function using arguments from each of the iterables. Stops when the shortest iterable is exhausted.
<code>max(iterable, *[, default=obj, key=func])</code>	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.

<code>max(arg1, arg2, *args, *[, key=func])</code>	With two or more arguments, return the largest argument.
<code>memoryview(obj)</code>	Create a new memoryview object which references the given object.
<code>min(iterable, *[, default=obj, key=func])</code>	→ smallest argument
<code>next(iter [,default])</code>	→ next item from the iterator. If default is given and the iterator is exhausted, it is returned instead of raising <code>StopIteration</code> .
<code>object()</code>	→ new featureless object. <code>object</code> is a base for all classes.
<code>oct(number)</code>	→ octal representation of an integer.
<code>open(file, **kwargs)</code>	Open file and return a stream (file object). Raise <code>IOError</code> upon failure. <code>open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)</code>
<code>ord(c)</code>	→ octal representation of an integer
<code>pow(x, y [,z])</code>	With two arguments, equivalent to <code>x**y</code> . With three arguments, equivalent to <code>(x**y) % z</code> , but may be more efficient (e.g. for ints).
<code>print(obj, ..., sep, end, file, flush)</code>	Prints the values to a stream, or to <code>sys.stdout</code> by default.
<code>property(**kwargs)</code>	Define managed attributes. See also decorator <code>@property</code>
<code>range(stop)</code>	→ sequence of numbers from 0 to stop - 1
<code>range(start, stop [,step])</code>	→ sequence of numbers from start to stop - 1 by step.
<code>repr(obj)</code>	→ canonical string representation of the object. For most objects <code>eval(repr(object)) == object</code>
<code>reversed(sequence)</code>	→ reverse iterator
<code>round(x [,ndigits])</code>	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. <code>ndigits</code> may be negative.
<code>set(iter)</code>	Build an unordered collection of unique elements.
<code>setattr(obj, name, val)</code>	Set a named attribute on an object; <code>setattr(x, 'y', v)</code> is equivalent to <code>x.y = v</code> .
<code>slice(start, stop [,step])</code>	Create a slice object. This is used for extended slicing (e.g. <code>a[0:10:2]</code>).
<code>sorted(iterable, key=None, reverse=False)</code>	→ new sorted list
<code>staticmethod(function)</code>	Convert a function to be a static method
<code>str([bytes-or-buffer [,enc [,err]]])</code>	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 <code>object.__str__()</code> (if defined) or <code>repr(object)</code> .
<code>sum(iter [,start])</code>	→ 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.
<code>super([type [,obj-or-type]])</code>	
<code>tuple([iter])</code>	→ tuple initialized from iterable's items
<code>type(obj)</code>	→ the obj's type
<code>vars([obj])</code>	→ dictionary; Without arguments, equivalent to <code>locals()</code> . With an argument, equivalent to <code>object.__dict__</code> .
<code>zip(*iters)</code>	→ a zip object whose <code>.__next__()</code> method returns a tuple where the i-th element comes from the i-th iterable argument. The <code>.__next__()</code> method continues until the shortest iterable in the argument sequence is exhausted and then it raises <code>StopIteration</code> .

Lit.:

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

CONTROL STRUCTURES

`if ...:` `for ...:`

```

        <block>
elif ...:
    <block>
else:
    <block>

        <block>

break

while ...:
    <block>
else:
    <block>

yield

```

FILES

```

fo.buffer()      fo.errors        fo.mode          fo.readline()    fo.truncate()
fo.close()       fo.fileeno()     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'):      (2)  with open('file.txt', 'r') as fo:
      for line in fo:                  lines = fo.readlines()
      ...                               ...
      fo.close()
```

CLASSES AND OBJECTS

Standard methods and operator redefinition in classes

Basic customization

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

Rich comparison

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

Attribute access

<code>__getattr__(self, name)</code>	Called when an attribute access has not found the attribute
<code>__getattribute__(self, name)</code>	Called unconditionally to implement attribute accesses for instances of the class.
<code>__setattr__(self, name, value)</code>	Called when an attribute assignment is attempted.

<code>__delattr__(self, name)</code>	Like <code>__setattr__()</code> but for attribute deletion instead of assignment.
<code>__dir__(self)</code>	Called when <code>dir()</code> is called on the object.
Implementing Descriptors	
<code>__get__(self, instance, owner)</code>	Get attribute
<code>__set__(self, instance, value)</code>	Set attribute
<code>__delete__(self, instance)</code>	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!

```

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>

<code>__future__</code>	Future statement definitions
<code>__main__</code>	The environment where the top-level script is run.
<code>_dummy_thread</code>	Drop-in replacement for the <code>_thread</code> module.
<code>_thread</code>	Low-level threading API.
<code>abc</code>	Abstract base classes according to PEP 3119.
<code>aifc</code>	Read and write audio files in AIFF or AIFC format.
<code>argparse</code>	Command-line option and argument parsing library.
<code>array</code>	Space efficient arrays of uniformly typed numeric values.
<code>ast</code>	Abstract Syntax Tree classes and manipulation.
<code>asynchat</code>	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.
 cgiitb 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.
 msvcr (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.
 pycldr 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()
christmas = datetime.date(2016, 12, 24)
print(christmas - today)    # 299 days, 0:00:00
dT = datetime.timedelta(days=8)
print(christmas + dT)      # 2017-01-01

```

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

Module operators

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

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

Example:

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

Module re

`import re`

Function	Description
<code>m = re.match(pattern, string, flags=0)</code>	Try to apply the pattern at the start of the string, returning a match object, or <code>None</code> if no match was found.
<code>re.findall(pattern, string, flags=0)</code>	Return a list of all non-overlapping matches in the string.
<code>m.group([group1, ...])</code>	→ str or tuple. Return subgroup(s) of the match by indices or names. For 0 returns the entire match.
<code>m.groups([default=None])</code>	→ 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
<code>time()</code>	→ time in seconds since the epoch as a floating point number, e.g. 1456693297.0732343
<code>asctime()</code>	→ e.g. 'Sun Feb 28 21:59:18 2016'
<code>localtime([secs])</code>	→ a <code>struct_time</code> object, e.g. <code>tm_year=2016</code> , <code>tm_mon=2</code> , <code>tm_mday=28</code> , <code>tm_hour=22</code> , <code>tm_min=3</code> , <code>tm_sec=59</code> , <code>tm_wday=6</code> , <code>tm_yday=59</code> , <code>tm_isdst=0</code> .
<code>strftime(format[, t])</code>	→ Convert a tuple or <code>struct_time</code> representing a time as returned by <code>gmtime()</code> or <code>localtime()</code> 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.pyplot 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>
- https://matplotlib.org/faq/usage_faq.html

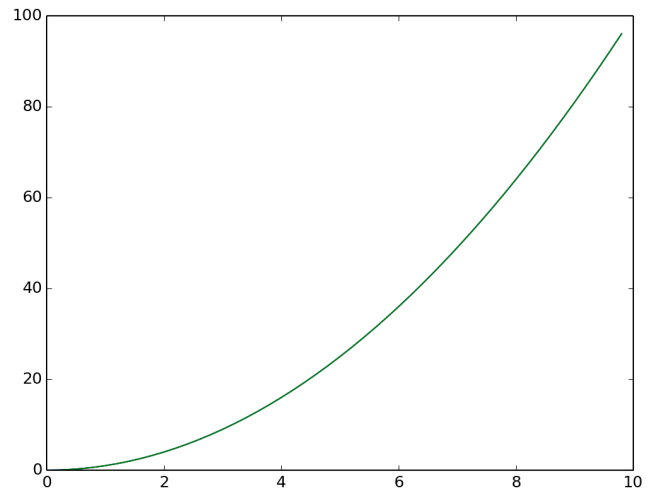
- SciPy

```
>>> 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
```



TOOLS

Package Distribution

- 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
 - 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 package
...
```

- Trick to choose Python version:

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

- Use twine to interact with PyPI.
- <http://www.pyinstaller.org> freezes programs into standalone executables on Windows.

Lit.: [9]

IPython

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

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

Press TAB key to auto-complete.

Commandline arguments

```
ipython --help
ipython locate           # ~/.ipython/
ipython profile help
ipython profile list
ipython profile create <name>
ipython --pylab          # Using matplotlib backend: Qt5Agg (default)
ipython --pylab=tk       # Using matplotlib backend: tk
ipython --matplotlib tk
ipython --matplotlib qt
ipython --profile=hubert  # Start ipython with profile "hubert"
```

```
Module IPython: import IPython; print(IPython.version_info)
Default config file  ~/.ipython/profile_default/ipython_config.py
Put startup code here: c.InteractiveShellApp.exec_lines = "..."
```

Ipython Magic

Enter `quickref` or `lsmagic` for a list of magic commands.

`%quickref` → IPy quick reference

`<magic>? → help`

`!ls` → system command; `L = !ls` store output in list `L`

In Prompt: In [26]: `_i26`, `_ih[26]`, `_ih[26:27]`

Out Prompt: Out [26]: `_26`, `_oh[26]`

Last 3 Inputs `_i`, `_ii`, `_iii`

Last 3 Outputs `_`, `---`, `---`

`##%` → cell

`%<magic>` → line magic; `%%<magic>` → cell magic

`%bookmark`

`%cd-<nr>`

`%cd ~`

`%clear` → Konsole löschen

`%dhist`

`%dirs`, `%pushd`, `%popd` → directory stack

`%edit`, `%ed` start editor (env var `ED-`

`TOR`)

`%env`

`%exec _i4`

`%exit`

`%history`, `%hist`

`%load_ext [extension]`

Example: automatic module reload

In [1]: `load_ext autoreload`

In [2]: `autoreload 2`

`%logstart [logfile] create log`
(def. `ipython_log.py`) in cwd

`%logstop` close log

`%logon` temporary on

`%logoff` temporary off

`%lsmagic` → list available magics

`%matplotlib` query mpl backend

`%matplotlib --list` show mpl backends

`%matplotlib gui` set mpl backend to gui

`%paste`, `cpaste`

`%pdb` Control debugger activation

`%prun` Python code profiler

`%pycat`

`%pylab`

`%pwd`

`%reset` → reset namespace

`%save`

`%rep _i4` → edit input line

`%run pgm.py` → run program

`%run -t pgm.py` → report timing

`%run -d -b<line> pgm.py` → debug, set breakpoint at line

`%run -d -b <file.py>:<line> pgm.py`

`%time`, `%timeit`

`%who`, `%who_ls`, `whos`

`%xdel` delete a variable

`%xmode` exc handler mode

Jupyter

`jupyter --help`

`jupyter --paths`

`jupyter <subcommand> --help`

`jupyter qtconsole --generate-config` # `~/jupyter/jupyter-qtconsole_config.py`

`jupyter notebook` # start notebook server

`jupyter nbconvert`

Edit `jupyter-qtconsole_config.py` to change settings, e.g. font size.

`jupyter nbconvert` can export to several formats, e.g. asciidoc, html, latex, markdown, rst, pdf.

Jupyter notebook keyboard 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	O,O

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-/

Conda/Pip Package Management

-----Conda-----	-----Pip-----
conda	
conda search <package>	pip search <package>
conda install <package>	pip install <package>
	pip install -U <package> # Upgrade
conda install python=x.x	
conda install --channel <name> <module>	
conda install --name env scipy	
conda list --name <env>	pip list
conda list --export	pip freeze
conda install python=x.x	
conda update python *	
conda install pip	pip install conda
conda remove <package>	pip uninstall <package>
conda remove --name <env> <package>	
	pip show conda

Lit.: <https://conda.io/projects/conda/en/latest/user-guide/tasks/>

Conda Misc

Anaconda Prompt Starter (Windows)

```
conda update menuinst
conda install -f console_shortcut
```

Install Mingw Compiler (only for external programs)

```
conda install -c anaconda mingw
```

Lit.: [10]

Virtual Environments

— Venv Environments —

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

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

Delete a virtual env by deactivating and removing the directory.

— Conda Environments —

```
-----Linux-----
conda create --name <env> pgm[=rev] ...
conda info --envs (list all envs)
source activate <env>
new: conda activate <env>
source deactivate
new: conda deactivate
conda remove --name myenv --all

conda list -e > pkgfile
conda create -n <env> --file < pkgfile

conda env --help # modern env
conda env export > pkgfile
conda env create -f=pkgfile -n <name>
```

```
-----Windows-----
activate <env>
deactivate
```

Example: Create environment testenv (Linux)

```
$ conda create --name testenv python=3.6
$ conda activate testenv
(testenv) hhoegl@e3:~$ _ # ~/miniconda3/envs/testenv/
```

Example: Create environment env_full with all pkgs in Anaconda (Linux)

```
$ conda create --name env_full anaconda
```

See pkgs at <https://docs.anaconda.com/anaconda/packages/pkg-docs/>

Lit.: [10]

ipdb

- \$ pip install ipdb
 - \$ python pgm.py
- ```
- pgm.py -
import ipdb
...
ipdb.set_trace()
```
- ipdb pgm.py # or ipdb3
  - ipython --pdb pgm.py

ipdb> help

Documented commands (type help <topic>):

```
=====
EOF cl disable interact next psource rv unt
a clear display j p q s until
alias commands down jump pdef quit source up
args condition enable l pdoc r step w
b cont exit list pfile restart tbreak whatis
break continue h ll pinfo return u where
bt d help longlist pinfo2 retval unalias
c debug ignore n pp run undisplay
```

## Spyder

### 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 |         |

### Windows Keys

|             |                     |
|-------------|---------------------|
| Ctrl-Sh-ESC | Task Manager        |
| Win-R       | Programm ausführen  |
| Win-Pause   | Systemeigenschaften |
| Win-E       | Explorer            |
| Win-D       | Desktop             |
| Ctrl-F      | Search              |

### Firefox Keys

|          |                    |
|----------|--------------------|
| C-0      | default font       |
| F5, C-R  | page reload        |
| /        | find text          |
| F1       | help               |
| F11      | toggle full screen |
| C-T      | new tab with focus |
| C-O      | open file          |
| C-L      | open location      |
| C-P      | open print dialog  |
| C-W      | close tab          |
| C-tab    | next tab           |
| C-Sh-tab | prev tab           |
| C-U      | page src           |

## References

- [1] Non-Programmer's Tutorial for Python 3  
[https://en.wikibooks.org/wiki/Non-Programmer%27s\\_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] Conda Documentation  
<https://conda.io/projects/conda/en/latest/index.html>



Your notes:

Your notes: