Summarised Python

Topics are in alphabetical order except for the first, new page for each:

Essentials
Bytes and Bytearray
Dictionaries
Exceptions or Errors
Files
Formatting
Functions
Lists
Mathematical
Random
Sets
Statements
Strings
Tuples
Value and Operators

```
print(2 * 3)
print("answers", 2 * 3, "and", 3 * 4)
print("answer", 2 * 3, end = ": ")
print("answers: ", 2 * 3, ", and ", 3 * 4, sep = "")

type(123)           (produces <class, 'int'>)
type(123) == int    (produces True)

12345   -7    3     int values
2.57    -3.1e+12    float values
"hello" "x"   ""    string values
'hello' 'x'   ''    same
True    False
None                like the probably familiar NULL or nil but more general

a / b               always a float
a // b              always an int
a ** b              to the power of

import library
from library import thing1, thing2, ...
from importlib import reload

x = 3

if conditionA:
  what to do if A true
elif conditionB:
  what to do if B true
else:
  what to do if all false

while condition:
  what to do while true

for variable in iterable:
  what to do for each value

def functionname(param, param, ...)
  what to do

break
continue
```

Bytes

immutable

```
a = b"Wake Up!"
a                                   # prints b'Wake Up!'
a[0]                        = 87    # 87 is ASCII code for W
as[0:3]                     = b'Wak'
bytes([87, 97, 108])        = b'Wak'
bytes(5)                    = b'\x00\x00\x00\x00\x00'   # i.e. 5 zero bytes
bytes("∑Æ.Ω", "utf8")       = b'\xe2\x88\x91\xc3\x86.\xce\xa9'   # unicode
"∑Æ.Ω".encode("utf8")       = b'\xe2\x88\x91\xc3\x86.\xce\xa9'   # same
b'\xe2\x88\x91\xc3\x86.\xce\xa9'.decode("utf8")
                            = '∑Æ.Ω'
b"Hello".decode()           = 'hello'  # ASCII is the default
```

Bytearray

mutable

```
a = bytearray(b"Wake Up!")
a                                 = bytearray(b'Wake Up!')
a = bytearray()                   = an empty one
a = bytearray(12)                 = length 21, values all zero
a = bytearray([65, 98, 99])   # ASCII codes equiv to bytearray(b"Abc")
a = bytearray("∑Æ.Ω", encoding = "utf8")
                              = bytearray(b'\xe2\x88\x91\xc3\x86.\xce\xa9')
a = bytearray(b"Wake Up!")
a[4] = "*"                        # an error
a[4] = 42                         # a is now bytearray(b'Wake*Up!')
a = bytearray(b"abcd")
a.append(65)                      # a is now bytearray(b'abcdA')
a.append(ord("A"))                # a is now bytearray(b'abcdAA')
a.extend([66, 67, 69])            # a is now bytearray(b'abcdAABCE')
a.pop()                           = 69  # a is now bytearray(b'abcdAABC')
a.extend(another bytearray)
a.insert(2, ord("-"))             # a is now bytearray(b'ab-cdAABC')
del a[3:6]                        # a is now bytearray(b'ab-AABC')
a.replace(bytearray(b"A"), bytearray("Rats"))
                                  # a is now bytearray(b'CBRatsRats-ba')
a.remove(ord("a"))                # a is now bytearray(b'CBRtsRats-ba')
b = bytearray(b"go ") + bytearray(b"away")
c = bytearray(b"123") * 4         # c is bytearray(b"123123123123")
     # += and *= also work
```

Dictionaries

`{}`	= an empty dictionary
`{ k1: v1, k2: v2, k3: v3, ... }`	= all `ki` must be hashable
`dict(k1 = v1, k2 = v2, k3 = v3, ... )`	if all `ki` look like variables
`D[ki]`	= `vi` error if key `ki` is not present in `D`
`D[ki] = vi`	= add new pairing or change existing one
`D[ki] += n`	update operators all allowed
`type(D) = dict`	= `True`
`D.keys()`	and others: a special object but:
`list[D.keys()]`	= a list of all the `ki`
`list[D.values()]`	= a list of all the `vi` in the same order
`list[D.items()]`	= a list of all the (`ki` , `vi`) tuples, same order
`len(D)`	= number of pairings it contains
`for i in D.keys()`	etc., all good
`for i in D.values()`	etc., all good
`for k in D:`	same as `in D.keys()`
`{ v: k for (k, v) in D.items() }`	comprehensions, this makes a reverse `dict`
`v in D`	= `True` or `False`, is `v` one of the `ki`?
`len(D)`	= number of `ki`: `vi` pairs
`del D[ki]`	remove the `ki`: `vi` pairing, error if not there
`D.pop(ki)`	same as `del D[ki]` but returns `vi`
`D.pop(ki, x)`	same but returns `x` if `ki` not present, no error
`D.popitem()`	removes and returns one `ki`: `vi` pair as a tuple
`D.update(otherD)`	other`D` is a dictionary, all its entries added to `D`
`sorted(D)`	= a list of all the `ki` only, in ascending order
`sorted(D, reverse = True)`	same as `sorted` but in descending order
`sorted(D, key = fn)`	the keys `ki` are sorted according to fn(`ki`)
`D1 | D2`	= new `dict` as if made by `D1.update(D2)`

Exceptions or Errors

Exceptions must be objects that inherit `BaseException`.

```
try:
   statements
except exceptiontype:
   statements
except (exceptiontype1, exceptiontype2, exceptiontype3, ...):
   statements
except exceptiontype as name:
   statements in which name is the exception object
except (exceptiontype1, exceptiontype1, exceptiontype1, ...) as name:
   statements in which name is the exception object
except:
   statements executed for any exception not explicitly mentioned
else:
   statements executed only if no exceptions were caught
finally:
   statements executed at the end no matter what

raise exceptionobject
raise                with no object inside except: re-reraises the caught exception


ex = BaseException(any number of arguments)

ex.args              = tuple of all parameters the exception's constructor got
ex.__traceback__     info on first function call that eventually led to the error
after etb = ex.__traceback__:
     etb.tb_frame.f_code.co_filename     = name of Python file
     etb.tb_frame.f_code.co_name         = function name or "<module>"
     etb.tb_lineno                       = the line number for the error
     etb = etb.tb_next    = None or same info on next call closer to the error

BaseException is inherited by
    AssertionError          assert statement failed
    AttributeError          accessing nonexistent x.y
    EOFError                attempt to read after end of file
    ImportError             import statement failed
    IndexError              accessing nonexistent x[y]
    MemoryError             run out of memory
    OSError                 failure in system call
    RecursionError          Python foolishly limits recursion depth
    RuntimeError            really just a general "miscellaneous"
    StopIteration           next() failed
    SyntaxError             error in Python code being read
    TypeError               wrong types for operation
    ValueError              a value is the right type but out of range
    ZeroDivisionError       what it says
```

```
f = open("filename", "r")
f = open("filename", "r", encoding = "utf8")
s = f.read()          # returns the entire file as a string with \n characters (1)
s = f.readlines()     # returns the entire file as list of strings each with \n (1)
s = f.readline()      # returns next line also with \n at end (1,2)
s = f.read(n)         # returns next n characters as string (2)
```

(1) last line of file has no \n if that's what's in the file
(2) no error at end of file, just shorter or empty string

```
for s in f:
```
loop in which s is each line of file in turn

```
f.close()

f = open("filename", "w")
f = open("filename", "w", encoding = "utf8")
f = open("filename", "a", ...)      # every write goes to end of file
print(a, b, c, file = f)
f.write(string)      # you must include \n if it is wanted

f.seek(0, 0)         # next read or write happens at beginning of file
f.seek(1234, 0)      # next read or write happens 1234 chars from beginning
f.seek(0, 1)         # return current position, measured in characters
f.tell()             # the same as f.seek(0, 1)
f.seek(0, 2)         # next write (or read) happens at end of file

f.open("filename", "r+", ...)   # read and write, file must exist
f.open("filename", "w+", ...)   # read and write, new file always created


f = open("filename", "rb", "wb", "ab", "rb+", or "wb+")   # binary file, then
f.write(b)      # b must be a bytes object
f.read()        # read entire file, return as bytes object
f.read(n)       # read next n bytes, return as bytes object
```
No other ways to read or write binary files.
The `seek`, `tell`, and `close` remain, positions measured in bytes not chars

```
f = open("name.csv", "r", newline = "")
csvr = csv.reader(f)
for row in csvr:
  ... # rows provided as lists of data items, one per line of the file
f.close()


f = open("name.csv", "w", newline = "")
csvw = csv.writer(f)
csv.writerow(L)                    # L's are lists of data items (all same length)
csv.writerows([L0, L1, L2, ...])   # each creating one line in the file

sys.stdout and sys.stdin        # keyboard and display as files
```

Formatting (1) the % operator

always string % tuple
or string % single value, both deliver a string result

```
"The square root of %d is %f" % (2, math.sqrt(2))
```
returns `'The square root of 2 is 1.414214'`

%a	(*1)	any object at all, formatted as for `print()`
%c		single character. parameter must be int or length=1 string
%d	(*2)	an int, printed in decimal
%E	(*3)	a float to be printed in "scientific" notation: `3.521E+03`
%e	(*3)	same as `%E` but `3.521e+03`
%F	(*3)	a float, never E notation, always just digits and a decimal point
%f	(*3)	exactly the same as `%F`, a float, just digits and decimal point
%G	(*3)	chooses between `%E` and `%F` for best appearance based on size
%g	(*3)	chooses between `%e` and `%f` for best appearance based on size
%i		same as `%d`, an int to be printed in decimal
%o	(*2)	an int to be printed in octal
%r	(*1)	any object, made printable with the `repr` function
%s	(*1)	any object, made printable with the `str` function
%X	(*2)	an int to be printed in hexadecimal, letters `ABCDEF`
%x	(*2)	an int to be printed in hexadecimal, letters `abcdef`
%%		just print `%`, no parameter is consumed

(*1) `%a, %r, %s`:
take any object at all, not just strings, and e.g.

%8s	minimum width 8 characters, spaces added at end if needed
%.24s	maximum width 24 characters, end cut off if needed
%8.24s	minimum width 8, maximum width 24 if needed
%-	- immediately after `%`, spaces added to beginning instead

(*2) `%d, %o, %X, %x`:

%8d	minimum width 8 characters, spaces added at *left* if needed
%08d	minimum width 8 characters, zeros added at left if needed
%-8d	minimum width 8 characters, spaces added at *right* if needed
%+8d	sign always shown even when positive
% 8d	(space) positive numbers are preceded by a space
%#...	for `%o, %X, %x` only, display `0o` or `0x` before the number

(*3) `%E, %e, %F, %f, %G, %g`:

%12f	minimum width 12 characters, spaces added at *left* if needed
%.8f	8 digits after the decimal point
%12.8f	min width 12 and 8 digits after the decimal point
%12.0f	don't even print the decimal point
- + 0	-, +, space, and 0 are the same as for (*2)
%#...	with all the above, decimal point always appears

Functions

```
def name(param, param, param, ...):
  body
```

In `def`'s parameter lists:

```
name = default

*                  # all subsequent params must be given with name = ...

* name             # name set to tuple of all unused parameters
                    # and all subsequent params require name = ...

* * name           # name set to dictionary of all remaining parameters
                    # which must be given as name = ...

def f(* a, * * b):  # a function that takes anything it is given,
                    # nameless ones in a, named ones in b
```

```
return
return value
```

Calls:

```
name()
name(value, value, ...)
name(name = value, name = value, ...)
name(value, value, ..., name = value, name = value, ...)
name(..., * dictionary, ...)     # entire contents become name = ... parameters
name(..., * iterable, ...)       # entire contents become separate parameters
```

Create a function object without giving it a name:

```
(lambda x: x + 1)(6)          = 7

(lambda x, y: x + y)(6, 8)    = 14
```

Closures:

```
def f(y):
  return lambda x: x + y

g = f(6)

g(8)                          = 14
```

```
a = [9, 3, "cat", 22, 3, 4, 1]
b = []
type(a) == list      = True
len(a)               = 7
a[0]                 = 9
a[-2]                = 4
a[3:6]               = [22, 3, 4]          # [3] is included, [6] is not
a[3:]                = [22, 3, 4, 1]
a[1:-3]              = [3, "cat", 22]
a.index(22)          = 3
a.count(3)           = 2
[5, 3] + [7, 1, 6]   = [5, 3, 7, 1, 6]
[2, 7] * 3           = [2, 7, 2, 7, 2, 7]
"cat" in a           = True
3 not in a           = False
tuple(a)             = (9, 3, "cat", 22, 3, 4, 1)
set(a)               = {9, 3, "cat", 22, 3, 4, 1}    # order varies
a[3] *= 2            # actually makes the change, unlike with tuples
a[2:5] = [99, 88]
a                    = [9, 3, 99, 88, 4, 1]
del a[-3:-1]
a                    = [9, 3, 99, 1]
del a[1]
a                    = [9, 99, 1]
a.append(5)
a                    = [9, 99, 1, 5]
a.append([2, 3, 4])
a                    = [9, 99, 1, 5, [2, 3, 4]]
a.extend([9, 8, 7])
a                    = [9, 99, 1, 5, [2, 3, 4], 9, 8, 7]
a.pop()              = 7
a                    = [9, 99, 1, 5, [2, 3, 4], 9, 8]
a.remove(99)                               # error if not present
a                    = [9, 1, 5, [2, 3, 4], 9, 8]
a.reverse()
a                    = [8, 9, [2, 3, 4], 5, 1, 9]
a.index(5)           = 2                   # error if not present
a.count(9)           = 2
a.clear()
a                    = []

b = [7, 3, 4, 2, 6]
c = [7, 3, 4, 2, 6]
def f(x): return 4 * x - x * x
[x * 3 for x in b]                    = [21, 9, 12, 6, 18]
[x / 2 for x in a if x > 3]           = [3.5, 2.0, 3.0]
[f(x) for x in b]                     = [-21, 3, 0, 4, -12]
sorted(b)                             = [2, 3, 4, 6, 7]
sorted(b, reverse = True)             = [7, 6, 4, 3, 2]
sorted(b, key = f)                    = [7, 6, 4, 3, 2]
b.sort()    # same options as sorted, but modifies b and returns nothing
```

continued ...

```
   min(b)              = 2
   max(b)              = 7

   b += [1, 9]
   b           = [7, 3, 4, 2, 6, 1, 9]
   b += (0, 5)
   b           = [7, 3, 4, 2, 6, 1, 9, 0, 5]

   j = [6, 3, 9]
   k = [7, 2, 3]
   l = [7, 2, 1]
   j < k          = True
   k < l          = False                    # same for >, >=, <=


x = [[1, 7], [5, 3]]
y = [[1, 7], [5, 3]]
z = x
w = copy.copy(x)
x == y                 = True
x == z                 = True
x == w                 = True
x is y                 = False
x is z                 = True
x is w                 = False
x[0] is w[0]           = True
v = copy.deepcopy(x)
x[0] is v[0]           = False

a = [ [0] * 6 ] * 6                          # 6x6 list of lists of zeros
a[1] is a[2]                           = True
a[1][3] = 6
a[2]                                   = [0, 0, 0, 6, 0, 0]
a = [ [0] * 6 for i in range(0, 6) ]         # 6x6 list of lists of zeros
a[1] is a[2]                           = False
a[1][3] = 6
a[2]                                   = [0, 0, 0, 0, 0, 0]
```

Mathematical

```
import math as m

m.pi            m.e             m.inf           m.nan

abs(x)          # not m.abs
m.fabs(x)       # same as abs but always a float

m.cos(x)        m.sin(x)        m.tan(x)
m.acos(x)       m.asin(x)       m.atan(x)
m.atan2(x, y)   # direction to (x, y) clockwise from North
m.cosh(x)       m.sinh(x)       m.tanh(x)
m.acosh(x)      m.asinh(x)      m.atanh(x)

m.degrees(r)    # convert radians to degrees
m.radians(d)    # convert degrees to radians

m.ceil(3.1) = 4                 m.ceil(-3.1) = -3
m.floor(3.1) = 3                m.floor(-3.1) = -4
m.trunc(3.1) = 3                m.trunc(-3.1) = -3
m.isclose(x, y, e)              # = m.fabs(x - y) <= e

m.dist((x1, y1), (x2, y2))      # pythagorean distance, any num of dims
m.hypot(x, y, z) = √(x²+y²+z²)  # any number of parameters

m.exp(x) = e^x                  m.exp2(x) = 2^x
m.log(x)        # natural base e m.log10(x)       m.log2(x)
m.sqrt(x)       m.cbrt(x)       m.pow(x, y)
m.isqrt(x)      # biggest int <= square root

m.gcd(a, b, c, ...)
m.lcm(a, b, c, ...)

m.factorial(x)
m.comb(n, r)    # combinations
m.perm(n, r)    # permutations

m.modf(73.185)  = (0.185, 73)   m.modf(-73.185) = (-0.185, -73)
m.remainder(x, y)               # x % y, works for floats
m.copysign(23, -75) = -23       m.copysign(23, 75) = 23
m.isfinite(x)   m.isinf(x)      m.isnan(x)
```

Random

`random.random()`	float >= 0 and < 1
`random.uniform(min, max)`	float in range
`random.randint(min, max)`	int in inclusive range
`random.gauss(mean, stddev)`	according to normal distribution
`random.getrandbits(N)`	= `random.randint(0, 2 ** N - 1)`
`random.randbytes(N)`	N element bytes object
`random.shuffle(list)`	no return, list is reordered randomly
`random.choice(list)`	pick one, all equally likely
`random.choices(list, wts)`	likeliness of wts[i] proportional to wts[i]
`random.choices(list, wts, k = N)`	list of N selected as above
`random.sample(list, N)`	any N items but each list[i] at most once

The following are good enough for cryptographic purposes:

`secrets.randbelow(N)`	int >= 0 and < N
`secrets.randbits(N)`	= `secrets.randbelow(2 ** N)`
`secrets.choice(list or tuple ...)`	pick one, all equally likely
`secrets.token_bytes(N)`	length N bytes object all 0 to 255
`secrets.token_hex(N)`	length 2 * N string of random hex digits

Sets

No duplicates. Adding something that is already there has no effect, not error.
May only contain hashable (usually = immutable) items.

`s = set()`	empty `{ }` makes a dictionary
`s = { 3, "cat", (5, 3) }`	
`type(s)`	`= <class, 'set'>`
`type(s) == set`	`= True`
`s.add(`item`)`	
`s.update(`tuple/list/set`)`	all members added individually
`s.update(`dictionary`)`	adds keys only
`s.discard(`item`)`	removes, OK if not present
`s.remove(`item`)`	removes, error if not present
`s.pop()`	removes and returns one item
`s.clear()`	remove everything
`sorted(s)`	a list of all contents in ascending order
`sorted(s, reverse = True)`	sorted but in descending order
`sorted(s, key =` fn`)`	the members e_i are sorted according to fn(e_i)

operators:

s_r = s_1 & s_2	intersection, set s_r = all that are in both s_1 and s_2
s_r = s_1 | s_2	union, set s_r = all that are in s_1 or s_2 or both
s_r = s_1 - s_2	set s_r = all that are in s_1 but not in s_2
s_r = s_1 ^ s_2	set s_r = all that are in s_1 or s_2 but not both
b = e in s	`True` or `False`, does `s` contain `e`?
b = e not in s	`True` or `False`, opposite of `in`
b = s_1 == s_2	`True` or `False`, do s_1 and s_2 have exactly the content?
b = s_1 != s_2	`True` or `False`, opposite of s_1 == s_2
b = s_1 <= s_2	subset, `True` or `False`, is everything in s_1 also in s_2?
b = s_1 < s_2	all in s_1 also in s_2 but something in s_2 is not in s_1
b = s_1 >= s_2	same as s_2 <= s_1
b = s_1 > s_2	same as s_2 <= s_1

Statements

```
variable = value

variable += value                          (and the other operators)

assert expression that must be true

break                                      (only in loops)

continue                                   (only in loops)

del unwanted thing, unwanted thing, ...

for variable in iterable:
   what to do for each value

global variable, variable, ...

if some condition:
   what to do if it's true
elif another condition:                    (optional and repeatable)
   what to do if that one's true
else:
   what to do if none of them are true

if condition: action                       (only for simple actions, not a good idea)
if condition: action, action, ...          (for all other structured statements too)

import module, module, ...
import module as abbreviation
from module import item, item, ...
from module import *                       (no name. needed for access)

pass

print(value, value, ...)                   ... file = x, sep = x, end = x

return value                               (only in functions)

while condition:
  what to do while it's true
else:                                      (optional)
  done after loop in not break
```

```
"xxx..."                any character except " and newline
'xxx...'                any character except ' and newline
"""xxx..."""            any characters except the sequence """
'''xxx...'''            any characters except the sequence '''

"abc"  "def"  "ghi"     same as "acbdefghi"
                        parentheses needed to span multiple lines

if type(s) == str:

str(12.56)              = "12.56"
str([9, False, ()])     = "[9, False, ()]"

"\u042f"                = "Я"      unicode characters in hexadecimal

len("abcdefghijklm")    13
"abcdefghijklm"[4]      "e"
"abcdefghijklm"[-3]     "k"
"abcdefghijklm"[4 : 7]  "efg"
"abcdefghijklm"[4 : -3] "efghij"
"abcdefghijklm"[4 : ]   "efghijklm"

"abc" + "def"           = "abcdef"
"abc" * 3               = "abcabcabc"
"fgh" in "abcdefghijklm" = True
"fh" in "abcdefghijklm"  = False

ord("A") = 65     chr(65) = "A"        by ASCII codes (actually unicode)

"abcdefghijklm".beginswith("abc")       = True
"abcdefghijklm".endswith("jklm")        = True
"abcdefghijklm".removeprefix("abc")     = "defghijklm"   (no change if
"abcdefghijklm".removesuffix("jklm")    = "abcdefghi"     not pre/suffix)
"CATxyzCAToooCATi".count("CAT")         = 3
"abcdXYZefghXYZij".find("XYZ")          = 4    (-1 if not present)
"abcdXYZefghXYZij".find("XYZ", 6)       = 11   (start search at position 6)
"abcdXYZefghXYZij".find("XYZ", 6, 9)    = -1   (end search at position 9)
"abcdXYZefghXYZij".rfind("XYZ")         = 11   (last occurrence, from start)
"abcdXYZefghXYZij".replace("XYZ", "##") = "abcd##efgh##ij'
"abcdXYZefghXYZij".partition("XYZ")     = ("abcd", "XYZ", "efghXYZij")
"abcdXYZefghXYZij".rpartition("XYZ")    = ("abcdXYZefgh", "XYZ", "ij")
"abcdXYZefghXYZij".partition("TT")      = ("abcdXYZefghXYZij", "", "")
"abc   de fgh i ".split()     = ["abc", "de", "fgh", "i"]
"abc   de fgh i ".split(" ")  = ["abc", "", "", "de", "fgh", "i", ""]
"CATxyzCAToooCATi".split("CAT")         = ["", "xyz", "ooo", "i"]
"   one two   three     ".strip()       = "one two   three"
"   one two   three     ".lstrip()      = "one two   three     "
"   one two   three     ".rstrip()      = "   one two   three"
"horse".center(11)       = "   horse   "
"horse".ljust(11)        = "horse      "
"horse".rjust(11)        = "      horse"
"horse".center(2)        = "horse"
```

continued ...

```
"horse".zfill(11)        = "000000horse"
"horse".center(11, ".")  = "...horse..."
"The cow said MOO!".upper()   = "THE COW SAID MOO!"
"The cow said MOO!".lower()   = "the cow said moo!"

"Once upon a Time".isalpha()  = False
"OnceuponaTime".isalpha()     = True
"".isalpha()                  = False
"O".isalpha()                 = True
```

Also:

`.isalnum()`	'a' to 'z' or 'A' to 'Z' or '0' to '9'
`.isascii()`	all codes between 0 and 127 inclusive
`.isdecimal()`	'0' to '9'
`.isalnum()`	'a' to 'z' or 'A' to 'Z' or '0' to '9'
`.isalnum()`	'a' to 'z' or 'A' to 'Z' or '0' to '9'
`.isidentifier()`	satisfies python rules for variable names
`.islower()`	'a' to 'z' only
`.isupper()`	'A' to 'Z'only
`.isspace()`	all white space: spaces, tabs. newlines, etc.
`.isprintable()`	anything visible, space included, tab and newline not

`s.expandtabs(N)`	replace all tabs with spaces so the result looks identical to s if tab stops are set every N character positions

Tuples

Immutable

Tuple of size 0: `()`
Tuple of size 1: `(a, )`
Tuple of size 2: `(a, b)`
Tuple of size 6: `(a, b, c, d, e, f)`

```
a = (9, 3, "cat", 22, 3, 4, 1)
type(a) == tuple    = True
len(a)              = 7
a[0]                = 9
a[-2]               = 4
a[3:6]              = (22, 3, 4)          # [3] is included, [6] is not
a[3:]               = (22, 3, 4, 1)
a[1:-3]             = (3, "cat", 22)
a.index(22)         = 3
a.count(3)          = 2
(5, 3) + (7, 1, 6)  = (5, 3, 7, 1, 6)
(2, 7) * 3          = (2, 7, 2, 7, 2, 7)
"cat" in a          = True
3 not in a          = False
list(a)             = [9, 3, "cat", 22, 3, 4, 1]
set(a)              = {9, 3, "cat", 22, 3, 4, 1}    # order varies

b = (7, 3, 4, 2, 6)
c = (7, 3, 4, 2, 6)
def f(x): return 4 * x - x * x
(x * 3 for x in b)                    = <generator object ...
tuple((x * 3 for x in b))             = (21, 9, 12, 6, 18)
tuple((x / 2 for x in a if x > 3))    = (3.5, 2.0, 3.0)
sorted(b)                             = [2, 3, 4, 6, 7]
sorted(b, reverse = True)             = [7, 6, 4, 3, 2]
tuple((f(x) for x in b))              = (-21, 3, 0, 4, -12)
sorted(b, key = f)                    = [7, 6, 4, 3, 2]

min(b)              = 2
max(b)              = 7

b += (1, 9)
b           = (7, 3, 4, 2, 6, 1, 9)      # immutable, did b = b + (1, 9)

j = (6, 3, 9)
k = (7, 2, 3)
l = (7, 2, 1)
j < k          = True
k < l          = False                   #  same for >, >=, <=
```

Values and Operators

Decimal:	`123`
	`0`
	`+45`
	`-9876`
Hexadecimal:	`0x1ab45`
	`0x1AB45`
Binary:	`0b1001011`
Octal:	`0o741`
Logical:	`True`
	`False`
null:	`None`

type-casts:
```
float(123)
int(3.9)        (truncates)
```

operators:
```
+  -  *  %  ==  !=  <  >  <=  >=
/           (result always an int)
//          (result always a float)
**          (to the power of)
&  |  ^  ~   (bitwise, only for ints)
<<  >>       (shifts, only for ints)
and or not   (only for logical)
```

if expression: `X if Y else Z`

constants, e.g.: `math.pi`

simple functions:
```
round(X)            (to nearest int)
round(X, N)         (to N digits after point)
abs(X)
min(X, Y, Z, ...)
max(X, Y, Z, ...)
```
library, e.g.: `math.sin(X)`

```
float("12.345")        = 12.345
float("  12.345  ")    = 12.345
int("-543")            = -543
int("100", 8)          = 64        (8 is base, anything from 2 to 36)
int("5A7F", 16)        = 23167
hex(23167)             = "0x5a7f"
```