Python Cheat Sheet

Data types

Types

Name	Python type	Examples
integer	`int`	`2; -45; 0`
floating point	`float`	`1.014; -12.64; 0.0`
string	`str`	`'b'; 'banana'; 'banana cake'`
boolean	`bool`	`True; False`
list	`list`	`[1, 2, 3]; [-1, 'a', True]`
tuple	`tuple`	`(1, 2, 3); (-1, 'a', True)`
dictionary	`dict`	`{'banana': 3, 'pear': 42, 'alien fruit': 0}`

Type functions

list

Function	Example	Explanation
`[]`	`fruits = ["apple", "grape", "kiwi]`	create an list of strings
`x.append(y)`	`[1, 2].append(3)` → `[1, 2, 3]`	add y to end of x
`x.pop(y)`	`[1, 2].pop(1)` → `2`	remove element at position y from x
`x.reverse()`	`[1, 2].reverse()` → `[2, 1]`	reverse order of x
`x.sort()`	`[1, 2, 0].sort()` → `[0, 1, 2]`	sort x
`x.count(y)`	`[1, 2, 1].count(1)` → `2`	count occurrences of y in x
`y in x`	`0 in [1, 2]` → `False`	check if y is in x
`x + y`	`[1, 2] + [3, 4]` → `[1, 2, 3, 4]`	concatenate x and y

string

Function	Example	Explanation
`""`	`name = "Rainer Zufall"`	create a string
`x.startswith(y)`	`'banana'.startswith('ban')` → `True`	check if x begins with y
`x.endswith(y)`	`'banana'.endswith('ana')` → `True`	check if x ends with y
`x.replace(y, z)`	`'banana'.replace('a', 'o')` → `'bonono'`	replace every y with z
`y in x`	`'anan' in 'banana'`→ `True`	check if y is in x
`x.join(y)`	`''.join(['First', 'Last'])` → `'First Last'`	combine elements in y with string x
`x.lower()`	`'BaNaNa'.lower()` → `'banana'`	convert x to lower case
`x.upper()`	`'BaNaNa'.upper()` → `'BANANA'`	convert x to upper case
`x.split(y)`	`'ba na na'.split('')` → `['ba', 'na', 'na']`	list of substrings in x separated by y
`x.strip()`	`'banana\n'.strip()` → `'banana'`	remove leading and trailing whitespace

`x + y`	`'ban'+ 'anana'`→ `'banana'`	concatenate x and y

dictionary

Function	Example	Explanation
`{}`	`id_2_name = {}`	create a new empty dictionary
`set`	`id_2_name['9606'] = 'Homo sapiens'`	add a new key value pair to the dict
`get`	`id_2_name['9606']` → `'Homo sapiens'`	access dictionary element using the key '9606'to get its value
`update`	`id_2_name['9606'] = 'Homo sapiens Linnaeus, 1758'`	overwrite existing value for given key
`delete`	`del id_2_name['9606']`	remove the entry with key '9606'

Slicing

Operation	Example	Explanation
`x[y]`	`[1, 2, 3, 4, 5][0]` → `1`	get single element
`x[y:z]`	`[1, 2, 3, 4, 5][1:3]` → `[2, 3]`	get elements from positions y to z-1
`x[y:]`	`[1, 2, 3, 4, 5][2:]` → `[3, 4, 5]`	get elements from positions y to the end
`x[y:-1]`	`[1, 2, 3, 4, 5][2:-1]` → `[3, 4]`	get elements from positions y to the second last

Operators

Arithmetic operators

Name	Operator	Example	Result
Addition	`+`	`9 + 2`	`11`
Substraction	`-`	`9 - 2`	`7`
Multiplication	`*`	`9 * 2`	`18`
Integer division	`//`	`9 // 2`	`4`
Float division	`/`	`9 / 2`	`4.5`
Exponent/Power	`**`	`9 ** 2`	`81`
Modulus	`%`	`9 % 2`	`1`

(*Note:* `+, -, *, / and ** have equivalents for direct assignment: +=, -=, *=, /= **=, e.g. x += 5 is equivalent to x = x + 5 )`

Boolean operators

Name	Operator	Example	Result
Equal	`==`	`9 == 2`	`False`
Not equal	`!=`	`9 != 2`	`True`
Larger	`>`	`9 > 2`	`True`
Larger equal	`>=`	`9 >= 2`	`True`
Smaller	`<`	`9 < 2`	`False`

Smaller equal	<=	9 <= 2	False
AND	and	(9 > 2) and (9 < 11)	True
NOT	not	not (9 > 2)	False
OR	or	(9 > 2) or (9 == 2)	True

Control flow (if, for, while)

If statement

```
if  9 > 2:
    print('9 bigger than 2' )
elif 9 < 2:
    print('9 smaller than 2')
else:
    print('9 equals 2')
```

For loop

```
for i in [1, 2, 3, 4]:
    print(i)
```

While loop

```
i = 1
while i <= 4:
    print(i)
    i += 1
```

Functions and modules

general

Function	Example	Explanation
`import x`	`import(math)`	import module x for usage
`from x import y`	`from math import sqrt`	import only x from module y
`print(x)`	`print('hello world')`	show x on screen
`help(x)`	`help(math)`	show documentation for x
`exit()`	`exit()`	exit python
`type(x)`	`type(myDNA)` → `string`	type of x
`len(x)`	`len([1, 2, 3, 4])` → `4`	length of x
`max(x, y)`	`max(1, 2)` → `2`	the larger of two objects (also: min())
`abs(x)`	`abs(-1)` → `1`	absolute value of x
`range(x, y)`	`range(1, 4)` → `[1, 2, 3]`	list over numbers from x to y-1
`round(x, y)`	`round(1.205, 2)` → `1.21`	round x to y decimals

math

Function	Example	Explanation
`math.sqrt(x)`	`math.sqrt(4)` → `2`	square root of x
`math.ceil(x)`	`math.ceil(0.9)` → `1.0`	round x up
`math.floor(x)`	`math.floor(0.9)` → `0.0`	round x down
`math.cos(x)`	`math.cos(0)` → `1.0`	cosine of x (analogous: math.tan(x), math.sin(x))

sys

Function	Example	Explanation
`sys.argv`	`['my_script.py' , 'my_dna.fasta']`	parameters with which the script called
`sys.path`	`['/Library/Python/2.7/site-packages']`	directories in which python searches for modules

re

Function	Example	Explanation

`re.search(exp, x)`	`re.search('xy', 'banana')` → None	check if regex exp occurs in string x
`re.search(exp, x).group(0)`	`re.search('ana', 'banana')` → 'ana'	check if exp in x and show pattern found
`re.findall(exp, x)`	`re.findall('ta', 'tautax')` → ['ta', 'ta']	find all occurences of exp in x
`[y|z]x`	'[A|a]lex'→ 'Alex'; 'alex'	hits substrings starting with y or z, followed by x
`xy*`	'TA*'→ 'T'; 'TA', 'TAA', ..	hits substrings starting with x followed by any number of y's
`\w`	'\wno'→ 'Ano'; 'zno', '9no', '_no', ..	hits any letter or number 0-9 (upper/lower case) and _
`\d`	'\d015'→ '0015'; '2015', ..	hits any number 0-9
`\s`	'Mr\s'→ 'Mr '; 'Mr\n', 'Mr\t'	hits any white space

Reading and writing files

Reading

```
#open file in reading mode
my_file = open('my_dna.fasta', 'r')

#read a single line
a_line = my_file.readline()

#iterate over all lines in a file
for line in my_file:
...
#read all lines
all_lines = my_file.readlines()

#close the file
my_file.close()
```

Writing

```
#open file in writing mode
my_file = open('my_dna.fasta', 'w')

#write string x into file
my_file.write(x)

#write all strings in list x into file
my_file.writelines(x)

#close the file
my_file.close()
```

IPython magics

Using other programming languages

```
R_results = %R <R command>
%html
%sql
%ruby
%cython
%python2
...
```

Profiling your code

`%timeit <python command>`
– returns average time your command took to run, useful when you want to optimize speed

`%prun/lprun -e <function name> <python command>`
– tells you the percentage of time your command spent in each line (lprun) or each function (prun), helps you narrow down performance critical parts in your code

Debugging

`%debug`
– enter pdb (python debugger), in which you can navigate and inspect variables that produced errors