Python 3 Cheat Sheet

EPFL CS 328
Numerical Methods for Visual Computing
(Version 2)

Basic data types and introspection

Basic native data types:

Integer	`i = 42`
Float	`3.14159`
Complex number	`2 + 3j`
Boolean	`b = True`
String	`s = 'spam'`
None type.	`n = None`

Introspection functions:

Type of an object	`type(var)`
Built-in system help	`help(var)`
Lists objetc's attributes	`dir(var)`
Class membership test	`isinstance(var, class)`

Operators

Arithmetic operators:

Addition	`x + y`
Subtraction	`x - y`
Floating point division	`x / y`
Integer division	`x // y`
Modulo	`x % y`
Multiplication	`x * y`
Exponentiation	`x ** y`

Boolean operators:

And	`x and y`
Or	`x or y`
Negation	`not x`

Printing and strings

Print statement:

```
print("Hello, world!")
print(3.14)
```

Formatting strings (f-strings):

```
my_float, my_string  = 1.2345, 'foo'
print(f"Float: {my_float}, String: {my_string}")
# Float: 1.2345, String: foo
print(f"Float: {my_float:.2f}")
# Float: 1.23
```

Lists

Ordered sequence of elements of arbitrary data types.

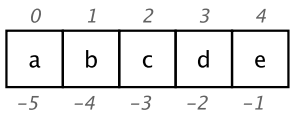
Create empty	`empty_l = []`
Create example	`l = ['zero', 1, 2.0, 3 + 0j]`
Retrieve item (idx from 0)	`d[2]  # Returns 2.0`
Change item	`l[2] = 'two_point_o'`
Query length	`len(l)  # Returns 4`
Append value to the end	`l.append(4)`
Extend by another list.	`l.extend([5, 5])`
# appearances of item.	`l.count(5) # Returns 2`

Looping through all items:

```
for it in l:
    # do something...
```

Slicing lists

```
a = ['a', 'b', 'c', 'd', 'e']
```



Syntax `[start:end]` (start - incl., end - excl., step=1)

Explicit start/end	`a[2:4]  # ['c','d']`
Implicit end (incl.)	`a[2:]  # ['c','d','e']`
Implicit start	`a[:3]  # ['a','d','e']`
Negactive indices	`a[1:-1] # ['b','c','d']`

Syntax `[start:end:step]`

Explicit start/end/step	`a[1:5:2]  # ['b','d']`
Negative step - backwards	`a[4:1:-2] # ['e','c']`
Implicit start/end/step=1	`a[::] # ['a','b','c','d','e']`
No valid index in range	`a[4:2:1]  # []`

Dictionaries

Mapping of key-value pairs.

Create empty	`empty_d = {}`
Create example	`d = {'name': 'Alice', 'age': 25}`
Retrieve entry	`d['age']  # Returns 25`
Add / change entry	`d['city'] = 'Lausanne'`
Delete entry	`del d['age']`
Delete all entries	`d.clear()`
Test if key exists	`'name' in d  # Returns True`
Number of entries	`len(d)`

Looping through all key-value pairs:

```
for key, val in d.items():
    # do something..
```

Similarly, access all keys or values as:

```
d.keys()
d.values()
```

Tuples

Immutable list of values.

Create empty	`t = ()`
Create with one element	`t = 123,  # Trailing comma`

Create example / packing	`t = 123, 'abc', 1+5j`
	`# Optional with parenthesis`
	`t = (123, 'abc', 1+5j)`

Unpacking	`u, v, w = t`
Unpacking some entries	`u, _, w = t`

Functions

Simple function:

```
def hello():
    print("Hello!")
```

Function with arguments and a return value:

```
def add(a, b):
    return a + b
```

Function with a default argument that has multiple return values as a tuple:

```
def f(a, b, c=0):
    return a + c, b + c
```

Conditional Statements

Conditional tests:

equal / not equal	`x == 25 , x != 25`
greater / smaller than	`x > 25 ,  x < 25`
greater /smaller or equal to	`x >= 25 , x <= 25`

If statement:

```
if x >= 0:
    print("Non-negative")
```

If-elif-else statement:

```
if x < 0:
    print("Negative")
elif x == 0:
    print("Zero")
else:
    print("Positive")
```

Loops

Use *for* to iterate over lists:

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

Otherwise, use *while* loops:

```
i = 0
while i < 3:
    print(x)
    i += 1
```

Cheat-sheet by T. Zeltner and J. Bednarik.
LaTeX template by Michelle Cristina de Sousa Baltazar.

List comprehensions

Syntax:

```
[expr(v) for v in some_list (if predicate(v))]
```

Get powers of 2 : $[2^0, 2^{10}]$:

```
l = [2**x for x in range(11)]
# [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

Get extension-less names of files with "jpg" extension:

```
files = ['img1.jpg', 'img2.png', 'img3.jpg']
l = [f[:-4] for f in files if f[-4:] == '.jpg']
# ['img1', 'img3']
```

Dict comprehensions

Syntax:

```
{expr(k):expr(v) for (k, v) in zip(l1, l2) (if predicate(k, v))}
{expr(k):expr(k) for k in some_list (if predicate(k))}
```

Create dict from 2 corresponding lists:

```
names = ['morty', 'rick']
ages = [14, 72]
db = {key:val for (key, val) in zip(names, ages)}
# {'morty': 14, 'rick': 72}
```

Map names to their lengths:

```
names = ['morty', 'rick']
db = {key:len(key) for key in names}
# {'morty': 5, 'rick': 4}
```

Importing modules

Import entire module:

```
>>> import math
>>> math.sqrt(2)
1.4142135623730951
```

Import specific functions:

```
>>> from math import sqrt
>>> sqrt(2)
1.4142135623730951
```

Giving a module (or functions) an alias:

```
>>> import math as m
>>> m.sqrt(2)
1.4142135623730951
```

Importing all functions from a module:
(Don't do this! It can result in naming conflicts.)

```
>>> from math import *
```