

Python 3 Cheat Sheet

EPFL CS 233

Introduction to Machine Learning

(Version 1)

Basic data types and introspection

Basic native data types:

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

Introspection functions:

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

Operators

Arithmetic operators:

Addition	<code>x + y</code>
Subtraction	<code>x - y</code>
Floating point division	<code>x / y</code>
Integer division	<code>x // y</code>
Multiplication	<code>x * y</code>
Exponentiation	<code>x ** y</code>

Boolean operators:

And	<code>x and y</code>
Or	<code>x or y</code>
Negation	<code>not x</code>

Printing and strings

Simple print statement:

```
print("Hello!")
```

String formatting:

Integers	<code>"int: %d" % 5</code>
Floats	<code>"float: %f" % 3.14</code>
Strings	<code>"str: %s" % "foo"</code>
Multiple values via tuples	<code>"two ints: %d %d" % (1, 2)</code>

Lists

Ordered sequence of elements of arbitrary data types.

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

Looping through all items:

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

Slicing lists

```
a = ['a', 'b', 'c', 'd', 'e']  
      0  1  2  3  4  5  ...  
      a  b  c  d  e  
... -6 -5 -4 -3 -2 -1
```

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

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

Syntax [start:end:step]

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

Dictionaries

Mapping of key-value pairs.

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

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	<code>t = ()</code>
Create with one element	<code>t = 123,</code> # Trailing comma
Create example / packing	<code>t = 123, 'abc', 1+5j</code> # Optional with parenthesis
Unpacking	<code>u, v, w = t</code>
Unpacking some entries	<code>u, _, w = t</code>

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	<code>x == 25</code> , <code>x != 25</code>
greater / smaller than	<code>x > 25</code> , <code>x < 25</code>
greater /smaller or equal to	<code>x >= 25</code> , <code>x <= 25</code>

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
```

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']
```

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 *
```