Cheatography

Python Cheat Sheet

by Reszi via cheatography.com/199671/cs/42210/

Lists

Add or remove	
Append	`my_lis t.a ppe nd( ele ment)`
Insert	`my_lis t.i nse rt( index, element)`
Remove by value	`my_lis t.r emo ve( ele ment)`
Remove by index	`del my_lis t[i ndex]` or `my_lis t.p op( inde x)`
Data Manipulation	
List compre-hension	`[expre ssion for item in list if condition ]`
Sorting	`my_lis t.s ort()`
Index of element	`my_lis t.i nde x(e lement)`
Count of element	`my_lis t.c oun t(e lement)`
Min/Max	`min/ma x(m ylist)`
Concat-enation	`list1 + list2`
List multip-lication	`[my_list] * n`
Condit-ional checks	`if all(logic)` or `if any(logic)`
Data Conversion	
List to string	`'separ ato r'.j oi n(m y_list)`

Lists (cont)

String to list	`string.sp lit ('s epa rat or')`
Shallow copy	`my_lis t.c opy()`
Deep copy (requires import copy)	`copy.d eep cop y(m y_list )`
Slicing	
Start, end, steps	`[start :en d:step]`
Reverse	`[::-1]`
Negative starts (from end)	`[-1:]`

Useful python

Writing files

```
with open(' fil ena me.t xt', 'w') as file:
    content = file.w rite()
```

Reading files

```
with open(' fil ena me.t xt', 'r') as file:
    content = file.r ead()
```

Checking type

```
if isinst ance(x, tuple):
```

Tuple unpacking

```
x, y = tuple(1, 2,)
```

Timing

```
import time
class Timer:
def __ente r__ (self): start_time = time.time()
def __exit __( self, *args) :en d_time = time.t im e()
print(f"{end_time - start_ time} seconds")
with Timer:
```

and vs *or*

`a = 1 b = 2`

`print(a or b)` returns a (first true or last false)

`print(a and b)` returns b (first false or last true)

Dictionary (copy)

Add or remove

Adding/Creating

```
mydict ['key'] = mydict  [' k e y'  ].g  et(v, defa
ult) + 1
```

Default dict (import defaultdict)

```
defaultdict(default_type)
```

Adding/Updating

```
my_dic t[' key3'] = 'value3'
```

Dictionary from two lists

```
dict(z ip( lis t_o f_keys, list_o  f_ v a lues))
```

Remove

```
del my_dic  t[ ' k ey1'] or
my_dic  t.p  o p(  'key2')
```

Accessing

Accessing

```
my_dic t[' key1'] or
my_dic  t.g  e t(  'key1', 'defau  lt _ v alue')
```

Keys list

```
list(my_dic t.k eys())
```

Values list

```
list(my_dic t.v alues())
```

Key: Value list

```
list(my_dic t.i tems())
```

Nested key/values

```
my_dic t[' out er_ key '][ 'in ner _key']
```

Safer nested dict

```
my_dic t.g et( 'ou ter _key', {}).ge  t( ' i nn  er
_  key')
```

Filter

```
{k: v for k, v in my_dic  t.i  t ems() if condition
}
```

Functions

Iterating

```
for key, value in my_dic  t.i  t ems():
```

Looping with enumerate

```
for index, (k, v) in enumer ate (my _di ct.i te ms(
)):
```

Dict comprehension

```
{k_e xpr : v_ exp r for item in iterable}
```

Dictionary (copy) (cont)

Merging

```
{dict1, dict2}
```

Sorting by key

```
dict(s ort ed( my_ dic t.i tem s()))
```

Sorting by value

```
dict(s ort ed( my_ dic t.i tems(), k=lambda item: it
m[1]))
```

Copying

Shallow copy

```
new_dict = my_dic  t.c  opy()
```

Deep copy (import copy)

```
new_dict = copy.d  ee p c op  y(m  y_ dict)
```