Python Cheat Sheet

Creator: Simon Zhang

List -> list() / [1, 2]

```
l1 = [1, 2, 3]
l2 = [4, 5]
l1.extend(l2)
l1.append(100)
l1.insert(0, 666)

l1.pop() # default pop last element, return it
l1.remove(2) # remove first accurance (value)
del l1[1] # remove at specific index

l1.reverse() # change in place, no return
l1.sort(min) # decresing order
l1.index(4) # first accurance
l1.count(0)
l1.sort(key=None, reverse=False) # change l1
sorted(l1) # l1 not change
```

String -> str

```
s = 'hello simon'
s.title() # 'Hello Simon'
len(s)
s[5:10]
s[-3:-1]
s.upper()
s.lower()
s.index('h')
s[0] == 'h'
s = '    Haha,   SSSimonnnnn   '
s.strip()
s.strip().split(',')
s.count('S')
s.find('XX') # -1 if not find
s.index('S') # throw error if not find
```

Dictionary -> dict() / {'one' : 1} / {'one'=1}

```
name = {'Simom' : 178,
        'Helen' : 165}
```

```
name['Simom']
name['Helen'] = 999
name['xxx'] # KeyError
name.get('xxx') # return None
name.get('Dudu', 30) # default value is 30
name.setdefault('Dudu', 30) # only set when key is not in dict

for key in name: # same as keys()
    print(key)

name['GuaiGuai'] = 25 # add / modify dict use dict[key] = value
del name['GuaiGuai']
name.pop('Helen', default=-1) # return value, remove entry
name.popitem() # return 2-tuple, LIFO order

name.keys()
name.values()
name.items()

# f-string, concatenate string and other type
for key, value in name.items():
    print(f'{key} height: {value}')
```

Tuple -> tuple() / (1,)

```
t1 = 1,
t2 = (1, 2, 3, 100, 5, 6)
t3 = tuple('hello') # ('h', 'e', 'l', 'l', 'o')
for t in t3:
    print(t)
max(t2)
sorted(t2)
len(t3)
```

Set -> set()

```
s = {c for c in 'abracaabroa' if c not in 'abc'} # {'r', 'o'}
s2 = set('foobar') # {'r', 'f', 'b', 'a', 'o'}
s3 = set(['a', 'b', 'foo'])
s.isdisjoint(s2)
s.issubset(s2)
s < s2
s.union(s2)
s.intersection(s2)
s2.difference(s) # s2 - s
s.symmetric_difference(s2) # either in s or s2, but not both
```

zip / enumerate

```
key = 'abc'
value = '123'
for pair in zip(key, value):
    print(pair)

list(enumerate('simon')) # [(0, 's'), (1, 'i'), (2, 'm'), (3, 'o'), (4,
'n')]
list(zip(range(5), 'simon'))
d = dict(enumerate('abc')) # {0: 'a', 1: 'b', 2: 'c'}
z = zip('xyz', [23, 24, 25])
dict(z)
d1 = {(0, 0) : 5}
```

stack

```
stack = list(range(6))
stack.append(10)
stack.append(20)
stack.pop()
```

queue

```
from collections import deque
queue = deque(['simon', 'helen', 'dudu'])
queue.append('neinei')
queue.popleft()
queue.popleft()
```

ASCII

```
print(ord('a'))    # 97
print(ord('A'))    # 65
print(ord('0'))    # 48

print(chr(97)) # 'a'
print(chr(65))  # 'A'
print(chr(48)) # '0'
```

Comprehensions & Generator

```
result = [mapping_expr for value in iterable if filter_expr]

result = [{'key': value} for value in iterable
        if a_long_filter_expression(value)]

result = [complicated_transform(x)
        for x in iterable if predicate(x)]

descriptive_name = [
    transform({'key': key, 'value': value}, color='black')
    for key, value in generate_iterable(some_input)
    if complicated_condition_is_met(key, value)
]

result = []
for x in range(10):
    for y in range(5):
        if x * y > 10:
            result.append((x, y))

return {x: complicated_transform(x)
        for x in long_generator_function(parameter)
        if x is not None}

squares_generator = (x**2 for x in range(10))

unique_names = {user.name for user in users if user is not None}

eat(jelly_bean for jelly_bean in jelly_beans
    if jelly_bean.color == 'black')
```

class

```
class Student:
    """This is a student class."""

    def __init__(self, name = "") -> None:
        self.name = name

    def set_name(self, name):
        self.name = name

    def get_name(self):
        return self.name

    def speak(self):
        print(f'Hello, World! I am {self.name}')

    def __private(self): # private
        print("NO")
```

```
    def public(self):
        print("YES")
        self.__private()

    def __repr__(self) -> str:
        return f'student name: {self.name}'

    def __str__(self) -> str:
        return '"informal" or nicely printable string representation of
object.'

stu = Student()
stu.set_name("Simon")
print(stu.__dict__) # dict of attributes
print(stu) # __str__
print(repr(stu)) # __repr__
print(stu.__doc__) # return """doc"""
```

decorator

```
def timer(fun):
    def inner():
        <inner_body>
        fun()
    return inner

import time

# no param
def timer(func):
    def inner():
        start = time.time()
        func()
        end = time.time()
        print('function {} cost {} seconds'.format(func.__name__,
round(end - start, 2)))
    return inner

# with params
def repeat_func(n):
    def wrapper(func):
        def inner():
            print('before function run')
            for i in range(n):
                func()
            print('after function run')
        return inner
    return wrapper
```