1 Data Structures

Lists, tuples, sets, dictionaries.

In [ ]: # instantiation

my_list = ["hello", "world"]
my_tuple = ("hello", "world")
my_set = set(["hello", "world"])
my_dictionary = {"hello": "world"}

In [ ]: # adding

my_list.append("how u doin")
my_tuple += ("how u doin", )
my_set.add("how u doin")
my_dictionary["how u"] = "doin"

In [ ]: # indexing and searching

print(my_list[0])
print(my_tuple[0])
print(my_dictionary["how u"])
print(list(my_set)[0])

# existence
print("hello" in my_set)
print("hello" in my_list)
print("hello" in my_tuple)
print("hello" in my_dictionary)

In [ ]: # removing

my_list.remove("how u doin")
my_set.remove("hello")
my_dictionary.pop("how u")

In [ ]: # sorting

my_list = sorted(my_list)

# slicing

new_list = [1, 2, 3, 4, 5, 6]
my_short_list = new_list[1:4]
new_list = new_list[::-1]

In [ ]: # length, concatenation, other operations

print(len(my_list))
print(len(my_set))
print(len(my_tuple))
print(len(my_dictionary))

print(my_list + ["how u", "doin"])
print(my_tuple + ("how u," "doin", ))

In [ ]: # iterating over list

for i in range(len(my_list)):
    print(my_list[i])

for element in my_list:
    print(element)

for element in list(my_set):
    print(element)

# list comprehensions
my_list = [1, 2, 3, 4]
transformed_list = [x * 2 for x in my_list]
print(transformed_list)
conditional_transformed_list = [x * 2 for x in my_list if x > 2]
print(conditional_transformed_list)

# iterating over dictionary
for key in my_dictionary:
    print(key, my_dictionary[key])

print(my_dictionary.items())

2 Functions

Returning values, arguments, keyword arguments, decorators, exception handling, args, kwargs.

In [ ]: # Doing something
def say_hello():
    print("hello world")

# Do nothing
def do_nothing():
    pass

In [ ]: # Taking arguments and returning a value
def add(a, b):
    return a + b

# Taking arguments and not returning a value
def print_string(s):
    print(s)

# Taking KEYWORD arguments
def subtract(x=0, y=0):
    return x - y

subtract(y=1, x=2)

In [ ]: # Inner functions
def do_something():
    def say_hello():
        print("hello world")

    say_hello()

do_something()

# Functions as first class citizens
print(do_something)

In [ ]: # Implementing a decorator using a wrapper
def my_decorator(func):
```python
def wrapper():
    print("Do something before function call!")
    func()
    print("Do something after function call!")
    return wrapper

def say_hello():
    print("hello world")

decorated_hello = my_decorator(say_hello)
decorated_hello()
```

```
In [ ]: # Syntactic sugar
@my_decorator
def say_hello():
    print("hello world")

say_hello()
```

```
In [ ]: # Handling exceptions
try:
    add(1, 2, 3)
except TypeError as error:
    print("Inputs don't match signature!")

try:
    print(0/0)
except:
    print("Math is hard!")
```

```
In [ ]: # Non-explicit argument definitions
def add(*args):
    total = 0
    for x in args:
        total += x
    return total

add(1, 2, 3)
```

```
# Non-explicit keyword argument definitions
def subtract(**kwargs):
    return kwargs['initial'] - kwargs['difference']

subtract(initial=2, difference=1)
```

### 3 Classes

Class definitions, magic methods.
In []: # Defining a class
class Sloth:
    def chill(self):
        print("The sloth is chilling.")

    def do_nothing(self):
        print("The sloth is doing nothing.")

sloth = Sloth()
sloth.chill()

In []: # Defining magic methods

class Sloth:
    def __init__(self, name):
        self.name = name

    def chill(self):
        print("The sloth is chilling.")

    def do_nothing(self):
        print("The sloth is doing nothing.")

    def say_name(self):
        print(self.name)

sloth = Sloth("Kevin")
sloth.say_name()

In []: class Sloth:
    def __init__(self, name):
        self.name = name

    def chill(self):
        print("The sloth is chilling.")

    def do_nothing(self):
        print("The sloth is doing nothing.")

    def say_name(self):
        print(self.name)

    def __eq__(self, other):
        if self.name == other.name:
            return True
        return False

sloth1 = Sloth("Kevin")
sloth2 = Sloth("Steve")
sloth3 = Sloth("Kevin")

print(sloth1 == sloth2)
print(sloth1 == sloth3)
print(sloth3 == sloth1)

In [ ]: class Sloth:
    def __init__(self, name):
        self.name = name

    def chill(self):
        print("The sloth is chilling.")

    def do_nothing(self):
        print("The sloth is doing nothing.")

    def say_name(self):
        print(self.name)

    def __eq__(self, other):
        if self.name == other.name:
            return True
        return False

    def __add__(self, other):
        self.name = "Mega Sloth"
        return self

sloth1 = Sloth("Kevin")
sloth2 = Sloth("Steve")
sloth3 = Sloth("Kevin")

print(sloth1 == sloth2)
print(sloth1 == sloth3)
print(sloth3 == sloth1)

sloth1 = sloth1 + sloth2
print(sloth1.name)