Functions

In [ ]:
```
def my_first_function():
    print('Hello world!')

print('type: {}'.format(my_first_function))

my_first_function()  # Calling a function
```

Arguments

In [ ]:
```
def greet_us(name1, name2):
    print('Hello {} and {}!'.format(name1, name2))

greet_us('John Doe', 'Superman')
```

In [ ]:
```
# Function with return value
def strip_and_lowercase(original):
    modified = original.strip().lower()
    return modified

uggly_string = '  MixED CaSe '
pretty = strip_and_lowercase(uggly_string)
print('pretty: {}'.format(pretty))
```

Keyword arguments

In [ ]:
```
def my_fancy_calculation(first, second, third):
    return first + second - third

print(my_fancy_calculation(3, 2, 1))

print(my_fancy_calculation(first=3, second=2, third=1))

# With keyword arguments you can mix the order
print(my_fancy_calculation(third=1, first=3, second=2))

# You can mix arguments and keyword arguments but you have to start with arguments
print(my_fancy_calculation(3, third=1, second=2))
```

Default arguments

In [ ]:
```
def create_person_info(name, age, job=None, salary=300):
    info = {'name': name, 'age': age, 'salary': salary}

    # Add 'job' key only if it's provided as parameter
    if job:
        info.update(dict(job=job))

    return info

person1 = create_person_info('John Doe', 82)  # use default values for job and salary
person2 = create_person_info('Lisa Doe', 22, 'hacker', 10000)
print(person1)
print(person2)
```

Don't use mutable objects as default arguments!

In [ ]:
```
def append_if_multiple_of_five(number, magical_list=[]):
    if number % 5 == 0:
        magical_list.append(number)
    return magical_list

print(append_if_multiple_of_five(100))
print(append_if_multiple_of_five(105))
print(append_if_multiple_of_five(123))
print(append_if_multiple_of_five(123, []))
print(append_if_multiple_of_five(123))
```

Here's how you can achieve desired behavior:

In [ ]:
```
def append_if_multiple_of_five(number, magical_list=None):
    if not magical_list:
        magical_list = []
    if number % 5 == 0:
        magical_list.append(number)
    return magical_list

print(append_if_multiple_of_five(100))
print(append_if_multiple_of_five(105))
print(append_if_multiple_of_five(123))
print(append_if_multiple_of_five(123, []))
print(append_if_multiple_of_five(123))
```

Docstrings

Strings for documenting your functions, methods, modules and variables.

In [ ]:
```
def print_sum(val1, val2):
    """Function which prints the sum of given arguments."""
    print('sum: {}'.format(val1 + val2))

print(help(print_sum))
```

In [ ]:
```
def calculate_sum(val1, val2):
    """This is a longer docstring defining also the args and the return value.

    Args:
        val1: The first parameter.
        val2: The second parameter.

    Returns:
        The sum of val1 and val2.

    """
    return val1 + val2

print(help(calculate_sum))
```

`pass` statement

`pass` is a statement which does nothing when it's executed. It can be used e.g. a as placeholder to make the code syntatically correct while sketching the functions and/or classes of your application. For example, the following is valid Python.

In [ ]:
```
def my_function(some_argument):
    pass

def my_other_function():
    pass
```