Functions

Python basics

Kunal Khurana

2023-10-06

Table of contents

Learning outcomes

- write functions and pass arguments
- how to use positional and keyword arguments, and how to use arbitrary number of arguments
- using functions with lists, dictionaries, if statements, and while loops.
- storing functions in seperate files called modules
- styling the functions so as they become structured and comprehensible

Remarks-

1. If you're writing a function and notice the function is doing too many different tasks, try to split the code into two functions.
2.

Arguments and parameters

```
# greet_user

def greet_user():
    print("Hello!")

#calling
greet_user()
```

```
Hello!
```

```
# passing information to the function

def greet_user(username):    #username would accept the value that i provide
    print(f"Hello, {username.title()}!")

# calling with username
greet_user('kunal')  #don't forget to call the function as string
```

```
Hello, Kunal!
```

```
# practice
def display_message(content):
    print(f"Hope you are feeling energized this monday, and you enjoyed your weekend well.

# calling
display_message("your prepration for the maths test!!")
```

```
Hope you are feeling energized this monday, and you enjoyed your weekend well. I'm curious al
```

```
# pets
def describe_pet(name, breed):        #no inverted commas here
    print(f"I've a {name.title()} at home!")
    print(f"{name.title()} belongs to {breed.title()} breed.")

# calling
```

```
describe_pet('bruno', 'german shephard')
```

```
I've a Bruno at home!
Bruno belongs to German Shephard breed.
```

```
describe_pet('ravi', 'husky')
```

```
I've a Ravi at home!
Ravi belongs to Husky breed.
```

```
### Default values for parameters
# pets
def describe_pet(name, breed = 'husky'):
    print(f"I've a {name.title()} at home!")
    print(f"{name.title()} belongs to {breed.title()} breed.")

# calling
describe_pet('bruno')  #by default the breed is husky!
```

```
I've a Bruno at home!
Bruno belongs to Husky breed.
```

Returning values

```
def get_formatted_name(first_name, middle_name, last_name):
    "Return a full name, neatly formatted."
    full_name = f"{first_name} {middle_name} {last_name}"
    return full_name.title()
titre = get_formatted_name ('hardy', 'singh' , 'sandhu')
print(titre)
```

```
Hardy Singh Sandhu
```

Returning dictionaries

```
def build_person(first_name, last_name, age = None):
    person = {'first' : first_name, 'last' : last_name}
    if age:                              # assigns a particular age to the group
        person['age'] = age
    return person

titre = build_person('jimi', 'hendrix', age = 27)
print(titre)
```

```
{'first': 'jimi', 'last': 'hendrix', 'age': 27}
```

Using a function with a while loop

```
def get_formatted_name(first_name, last_name):
    "Return a full name, neatly formatted."
    full_name = f"{first_name} {last_name}"
    return full_name.title()

# infinite loop
while True:
    print("\nPlease tell me your name:")
    f_name = input("First name: ")
    l_name = input("Last name: ")

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")
```

```
def get_formatted_name(first_name, last_name):
    "Return a full name, neatly formatted."
    full_name = f"{first_name} {last_name}"
    return full_name.title()

# adding break to stop the loop
while True:
    print("\nPlease tell me your name:")
    print("(enter 'q' at any time to qutit)")
```

```
    f_name = input("First name: ")
    if f_name == 'q':
        break

    l_name = input("Last name: ")
    if l_name == 'q':
        break

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")
```

```
Please tell me your name:
(enter 'q' at any time to qutit)
First name: Kunal
Last name: Khurana

Hello, Kunal Khurana!

Please tell me your name:
(enter 'q' at any time to qutit)
First name: q
```

```
# capital, country

def get_capital_country (capital, country):
    cap_c = f"{capital} {country}"
    return cap_C.title()

while True:
    print("\nPlease print capital name and country name)
    print("enter q anytime to quit")

    capital = input("Capital name : ")
    if capital == 'q':
          break

    country = input("Country name : ")
    if country == 'q':
          break
```

```
# city_country
def city_country (city, country):
    formatted_string = f"{city.title()}, {country.title()}"
    return formatted_string
# INFINITE LOOPL; HOW TO STOP

count = 0
while count < 3:
    print("\nThe name of the city and capital are-")
    print(city_country('newyork', 'united states'))
    print(city_country('delhi', 'india'))
    count += 1 #increment the count

print ("loop has ended")
```

```
The name of the city and capital are-
Newyork, United States
Delhi, India

The name of the city and capital are-
Newyork, United States
Delhi, India

The name of the city and capital are-
Newyork, United States
Delhi, India
loop has ended
```

Modifying the list in a function

```
def print_models(unprinted_designs, completed_models):  #defining with two parameters
    while unprinted_designs:
        current_design = unprinted_designs.pop()
        print(f"Printing model: {current_design}")
        completed_models.append(current_design)

def show_completed_models(completed_models):           #defining with one parameter
    print("\nThe following models have been printed:")
    for completed_model in completed_models:
        print(completed_model)
```

```
unprinted_designs = ['ferrari', 'lancer', 'accord']
completed_models = []

print_models(unprinted_designs, completed_models)     #don't forget to print
show_completed_models(completed_models)
```

```
Printing model: accord
Printing model: lancer
Printing model: ferrari

The following models have been printed:
accord
lancer
ferrari
```

```
print_models(unprinted_designs[:], completed_models)     #[:] will save a copy to the old
show_completed_models(completed_models)
```

```
The following models have been printed:
accord
lancer
ferrari
```

Passing an arbitrary number of arguments

```
def make_pizza(*toppings):
    print(toppings)#prints requested toppings

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

```
('pepperoni',)
('mushrooms', 'green peppers', 'extra cheese')
```

The asterisk in the parameter name *toppings tells Python to make an empty tuple called toppings and pack whatever values it receives into this tuple.

```python
def make_pizza(*toppings):
    print("\nMake the pizza with following toppings-")
    for topping in toppings:
        print(f"-{topping}")

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

```
Make the pizza with following toppings-
-pepperoni

Make the pizza with following toppings-
-mushrooms
-green peppers
-extra cheese
```

```python
### adding an additional paramter 'size'
def make_pizza(size, *toppings):
    print(f"\nMake a {size}-inch pizza with the following toppings-")
    for topping in toppings:
        print(f"-{topping}")

make_pizza(12, 'pepperoni')
make_pizza(16, 'mushrooms', 'green peppers', 'extra cheese')
```

```
Make a 12-inch pizza with the following toppings-
-pepperoni

Make a 16-inch pizza with the following toppings-
-mushrooms
-green peppers
-extra cheese
```

Using arbitrary keyword arguments

1. (**) before user_info allows the function to accept any number of keyword arguments and store them as key-value pairs in a dictionary called user_info.

```
def user_profile(first, last, **user_info):
    user_info['first_name'] = first
    user_info['last_name'] = last
    return user_info

more_info = user_profile('Kunal', 'Khurana',
                         location= 'montréal',
                         field= 'agirculture')

print(more_info)
```

```
{'location': 'montréal', 'field': 'agirculture', 'first_name': 'Kunal', 'last_name': 'Khurana
```

Storing functions in modules

1. module import- writing a code and storing it somewhere; simply importing it and using it. e.g.- from maths import stats
2. generally written as module_name.function_name()
3. may be used to import as many funcitons as needed, simply seperating them by comma. e.g.- from module_name import function_0, function_1, fucntion_2
4. an alias can be provided to make the process simpler e.g.- from pizza import make_pizza as mp. This would save us some time while writing the code
5. we may also import all the functions from modules also with asterik*() e.g.- from pizza import*