Introduction to Python for Economists

Session 3: User input, functions & classes

Roland Mühlenbernd

26. Februar 2020

User input via input() function

1. `message = input("Write something to repeat: ")`
2. `print(message)`

Save the file as slides030202_input.py and start it from the terminal

- The input() function takes an argument: the *prompt*.
- It waits until the user enters the response and presses ENTER.
- The response is assigned to the variable message.

1. `name = input("Please enter your name: ")`
2. `print(f"\nHello, {name}!")`

Prompt as a variable

1. `prompt = "We need some information from you."`
2. `prompt += "\nWhat is your first name? "`
3. `name = input(prompt)`
4. `print(f"\nHello, {name}!")`

Note: ‘s += t ’ adds adds a string t to the string s.

Accepting numerical input

1. `age = input("How old are you? ")`
2. `print(age)`
3. `age = int(age)`
4. `print(age >= 18)`

Input and while loop

```
1. prompt = "\nTell me something to repeat."
2. prompt += "\nEnter 'x' to end the program. "
3. message = ""
4. while message != 'x':
5.     message = input(prompt)
6.     print(message)
```

The program prints ‘x’ as it would be a normal message. How can we avoid it?

Using a flag variable

```
1. prompt = "\nTell me something to repeat."
2. prompt += "\nEnter 'x' to end the program. "
3. active = True
4. while active:
5.     message = input(prompt)
6.     if message == 'x':
7.         active = False
8.     else:
9.         print(message)
```

Note: The flag active can be set to False for different reasons.

Using the break statement

```
1. prompt = "\nEnter cities you have visited."
2. prompt += "\nEnter 'x' to end the program. "
3. while True:
4.     city = input(prompt)
5.     if city == 'x':
6.         break
7.     else:
8.         print(f"I loved visiting {city.title()}!")
```

Exercise VII

- Write a program that asks the user what kind of car they would like to rent, check if the car is in a list (that you created before) and return the answer, either ‘Yes we have a BMW left’, or ‘I am sorry, we are out of BMWs today.’
- A movie theater charges different tickets prices depending on a person’s age. If a person is under 3, the ticket is free; if they are between 3 and 12, the ticket is 10 Euros; and if they are over 12, the ticket is 15 Euros. Write a loop in which you ask users the age and then tell them the ticket value. The loop can be ended with ‘x’.

A greeting function

```
1. def greet_user():
2.     """Display a simple greeting."""
3.     print("Hello!")
4. ...
5. greet_user()
```

- keyword **def** informs Python that you define a function
- a function without input **parameter** has empty brackets
- triple quotes mark **docstrings**, which generates a documentation of a function

A greeting function

1. `def greet_user(username):`
2. `    """Display a simple greeting."""`
3. `    print(f"Hello, {username.title()}!")`
4. `...`
5. `greet_user('tim')`

- the function has the **parameter** username
- in this example 'tim' is an **argument**

A greeting function

```
1. def greet_two_users(username1, username2):
2.     """Display a simple greeting."""
3.     print(f"Hello, {username1.title()}!")
4.     print(f"Hello, {username2.title()}!")
5. ...
6. greet_two_users('tim', 'sarah')
```

- the function has the two **parameters** and needs two **arguments**

A greeting function

```
1. def greet_users(usernames):
2.     """Display a simple greeting."""
3.     for username in usernames:
4.         print(f"Hello, {username.title()}!")
5. ...
6. greet_users(['tim', 'sarah', 'kai', 'max'])
7. users = ['tim', 'sarah', 'kai', 'max']
8. greet_users(users)
```

- the function needs a list as an **argument**

Order matters

1. `def owner_relationship(name, animal):`
2. `    """Display a owner animal relationship."""`
3. `    print(f"{name.title()} owns a {animal.title()}.")`
4. `...`
5. `owner_relationship('tim', 'dog')`
6. `owner_relationship('cat', 'sarah')`

- the order of arguments is produced by the order of parameters

Default and keywords arguments

1. `def owner_relationship(name, animal = 'dog'):`
2. `    """Display a owner animal relationship."""`
3. `    print(f"{name.title()} owns a {animal.title()}.")`
4. `...`
5. `owner_relationship('tim')`
6. `owner_relationship('sarah', 'cat')`
7. `owner_relationship(name ='marc', animal = 'rabbit')`
8. `owner_relationship(animal = 'rabbit', name ='marc', )`

- 'dog' is a **default argument** for animal. By only naming one argument, the variable animal is takes the value 'dog'.
- Note: when you use **keyword arguments**, the order doesn't mater

Satisfy arguments

1. `def owner_relationship(name, animal = 'dog'):`
2. `    """Display a owner animal relationship."""`
3. `    print(f"{name.title()} owns a {animal.title()}.")`
4. `...`
5. `owner_relationship('tim')`
6. `owner_relationship('sarah', 'cat')`
7. `owner_relationship()`

- since the parameter name is not defined as a default argument, the function needs at least on argument to be satisfied

Order matters

1. `def owner_relationship(name = 'harry', animal):`
2. `    """Display a owner animal relationship."""`
3. `    print(f"{name.title()} owns a {animal.title()}.")`
4. `...`
5. `owner_relationship('dog')`
6. `owner_relationship('sarah', 'cat')`

- It is not allowed to have a parameter without default after a parameter with default value

Return values

```
1. def get_formatted_name(first_name, last_name):
2.     """Return a full name, neatly formatted."""
3.     full_name = f"{first_name} {last_name}"
4.     return full_name.title()
5. ...
6. musician = get_formatted_name('jimi', 'hendrix')
7. print(musician)
```

- For a function that returns a value, you need a variable that the return value has to be assigned to.

Return values

1.
```
def get_formatted_name(first_name, middle_name, last_name):
```
2.
```
    """Return a full name, neatly formatted."""
```
3.
```
    full_name = f"{first_name} {middle_name} {last_name}"
```
4.
```
    return full_name.title()
```
5.
```
...
```
6.
```
musician = get_formatted_name('john', 'lee', 'hooker')
```
7.
```
print(musician)
```
8.
```
musician = get_formatted_name('jimi', 'hendrix')
```

- How can we alter the function to satisfy it with both musicians' names?

Using a function with a while loop

```
5.  ...
6.  while True:
7.      print("\nPlease tell me your name:")
8.      print("(enter 'x' to quit)")
9.      f_name = input("First name: ")
10.     if f_name == 'x':
11.         break
12.     l_name = input("Last name: ")
13.     if l_name == 'x':
14.         break
15.     formatted_name = get_formatted_name(f_name, l_name)
16.     print(f"\Hello, {formatted_name}!")
```

Returning a Dictionary

```
1. def build_person(first_name, last_name):
2.     """Return a dictionary of person information."""
3.     person = {'first': first_name, 'last': last_name}
4.     return person
5. ...
6. musician = build_person('jimi', 'hendrix')
7. print(musician)
```

Returning a Dictionary

```
1. def build_person(first_name, last_name, age=None):
2.     """Return a dictionary of person information."""
3.     person = {'first': first_name, 'last': last_name}
4.     if age:
5.         person['age'] = age
6.     return person
7. ...
8. musician = build_person('jimi', 'hendrix', age=27)
9. print(musician)
```

- You can pass over much more arguments.
- You can use function to construct all dictionaries lists and complex combinations of both.

Editing a Dictionary

```
1. def edit_person(name_dict, first_name, last_name, age=None):
2.     """Edits a dictionary of person information."""
3.     name_dict['first'] = first_name
4.     name_dict['last'] = last_name
5.     if age:
6.         person['age'] = age
7.     return name_dict
8. ...
9. my_dict = {'first': 'Elvis', 'job': 'Musician'}
10. my_dict = edit_person(my_dict, 'jimi', 'hendrix', age=27)
11. print(my_dict)
```

Passing arbitrary number of arguments

1. `def make_pizza(*toppings):`
2. `    """Print a list of required toppings."""`
3. `    print(toppings)`
4. `...`
5. `make_pizza('tomatoes')`
6. `make_pizza('tomatoes', 'cheese', 'onions')`

- *toppings tells Python to create a tuple called toppings with all the arguments as values

Order matters

```
1. def make_pizza(size, *toppings):
2.     """Print a list of required toppings."""
3.     print(f"Make {size}-inch Pizza with: "
4.     for topping in toppings:
5.         print(f"- {topping}")
6. ...
7. make_pizza(16, 'tomatoes')
8. make_pizza(12, 'tomatoes', 'cheese', 'onions')
```

- Keyword arguments first, remaining arguments later

Using arbitrary keyword arguments

```
1. def build_profile(first, last, **user_info):
2.     """Build a dictionary of any person information."""
3.     user_info['first_name'] = first
4.     user_info['last_name'] = last
5.     return user_info
6. ...
7. user_profile = build_profile('albert', 'einstain',
8.                              location = 'princeton',
9.                              field = 'physics')
10. print(user_profile)
```

Storing your functions in modules

1. `def make_pizza(size, *toppings):`
2. `    """Print a list of required toppings."""`
3. `    print(f"Make {size}-inch Pizza with: "`
4. `    for topping in toppings:`
5. `        print(f"- {topping}")`

- Remove anything except the function
- Save the file as pizza_module.py
- Open a new file (same dir) and write the following:

1. `import pizza_module` or
   `from pizza_module import make_pizza`
2. `make_pizza(16, 'tomatoes')`
3. `make_pizza(12, 'tomatoes', 'cheese', 'onions')`

Storing your functions in modules

- Note: when you import multiple modules, you might get a conflict when different functions have the same name
- This can be solved with the **from...import...as** statement

1. `from pizza_module import make_pizza as mp`
2. `mp(16, 'tomatoes')`
3. `mp(12, 'tomatoes', 'cheese', 'onions')`

- you can also use an alias for the whole file/module:

1. `import pizza_module as pmod`
2. `pmod.make_pizza(16, 'tomatoes')`
3. `pmod.make_pizza(12, 'tomatoes', 'ham')`

Functions Styling Guide

- Descriptive names, lowercase and underscore
- Comments in docstring that describes what the function does
- If the number of arguments extends the width of a page ($>$ 79 characters), use multiple lines with ENTER and 2 TABs

```
function_name(
        parameter0, parameter1, parameter2,
        parameter3, parameter4, parameter5):
    function body...
```

- If you have multiple functions in one file, leave a two blank lines between them

Exercise VIII

- Write a function called describe_city() that accepts the name of a city and its country. It prints a simple sentence, such as ‘Reykjavik is in Iceland’. Give the parameter for the country a default value. Call your function for three different cities, of which at least one is not in the default country.
- Write a function add_city_to_dict that takes a dictionary a city and a country, and adds the city as key and the country as value to it, and return the new dictionary
- create an empty dictionary, make a list of five city-country tuples, loop through the list and add the pairs to the dictionary by calling the function add_city_to_dict
- run through the newly created dictionary and print the information woth teh function describe_city()
- comment your code for better readability
- move the two function to an external file city_module and import it to the current file

Room for notes on Classes

More room for notes

More room for notes