Cmpe 150 Lab 2: Functions

Last Week

- We learned about how computers work and how we can do basic operations like getting input, doing arithmetic, and giving some output.

- We can do several things with these tools, yet we ought to be careful.

Spaghetti Code

- We may need hundreds or even thousands of instructions/lines of code for complex tasks.

- It’s super hard to understand and maintain such code. Comments won’t be enough to solve the issue.

Solution: More Structured Code with Functions

- A function is a unit of program carrying out a specific task.
- To define a function, we use a syntax like the following. (With indentation)
- def function_name():

    your_code_line_1

    your_code_line_2

How to Use/Call a Function

- After we define it as described in the previous slide, we just say function_name() in our code.

- When we do this, our computer jumps to the part where we define what we want our function to do. Executes instructions there, and then it returns to where it jumped from.

How to Use/Call a Function

```
def my_first_function():
    print("Hello this is my first function.")
    print("I am printing some output by using it.")


my_first_function()
```

"Why is my function not outputting anything"?

"Oh I never called the function"

ProgrammerHumor.io

Main Function

- As a convention, we define a function called main, corresponding to the whole program.

- If we write everything inside the main, are we done?

Main Function Convention

```
def main():
        line1
        line2


if __name__ == '__main__':
    main()
```

Inside a function, we can call another function.

```
def main():
    function1()
    function1()


def function1():
    print('I am running function 1')


main()
```

We Can Even Call Multiple Functions

```
def main():
    function1()
    function2()
def function1():
    print('I am running function 1')
def function2():
    print('I am running function 2')
main()
```

Idea

- The main function will correspond to the whole program. We will call different functions inside it, each corresponding to a subtask. We'll define what each of these functions will do as well.

- Is it sufficient? Can we deal with 1000s of lines of code?

Idea (Cont.)

- Further, divide each new function into smaller components again until you reach a point where dividing a function does not make sense. For instance, calculating 2x+5 can be a single function—no need to divide.

- Also, it is better to stay within 5-6 lines of code in a function at most. Not as a restriction imposed by Python but as a good practice. So that everyone can understand your code more easily.

Bad Example

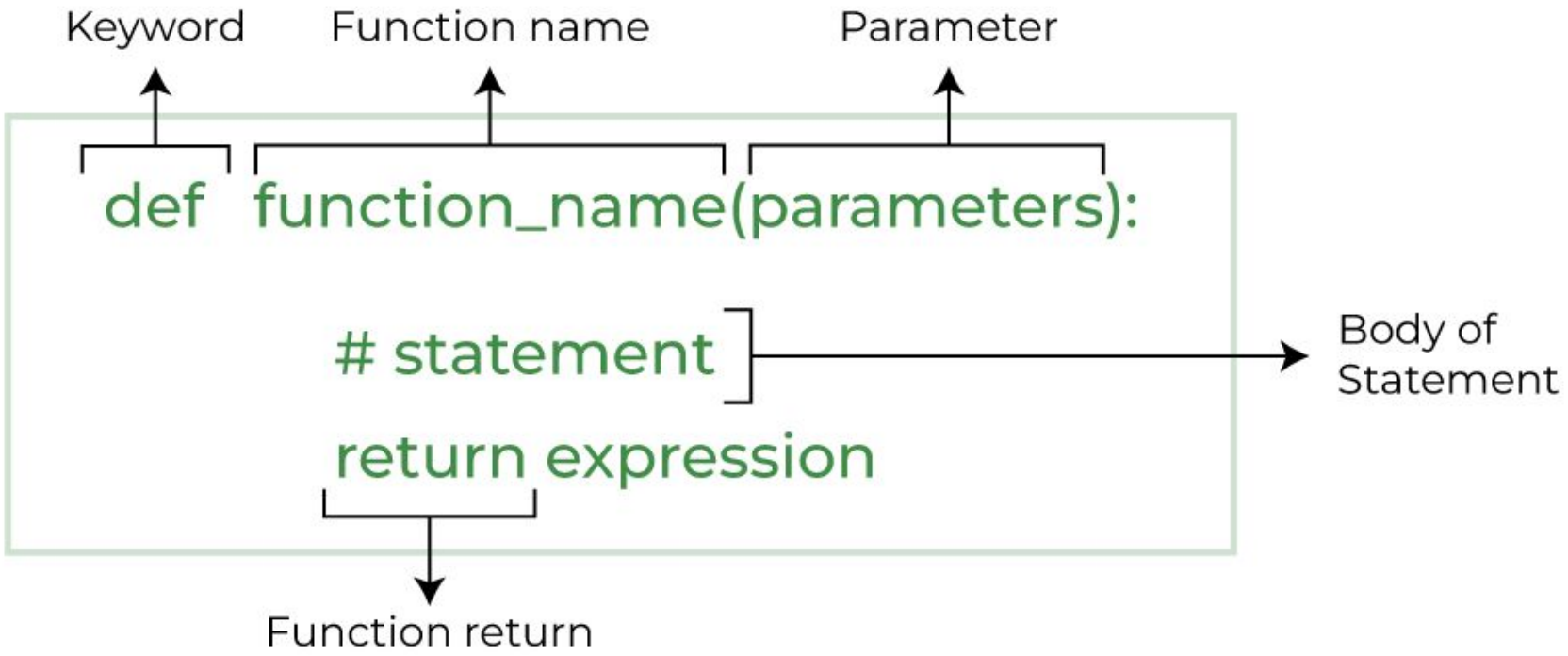
```
showChapter(chapter, tutorial) {
  if(chapter.orderIndex == 1) {
    this.redirectToRoot()
  }

  let higherChapters = chapter.items.filter(item => item.higher)
  let lowerTutorials = chapter.items.filter(item => !item.higher)

  let title = chapter.title
  let description = chapter.description

  if(chapter.lowerItem) {
    this.nextPath = this.getChapterPath(this.tutorial, chapter.lowerItem)
    this.nextLink = this.nextPath
  } else if(!!tutorial.pathId && !!lowerTutorials) {
    this.nextPath = this.getTutorialPath(lowerTutorials[0])
    this.nextLink = this.nextPath
  } else if (chapter.last && !tutorial.footLinks.length > 0) {
    this.nextPath = tutorial.footLinks.url
  }

  if(higherChapters.length === 0) {
    this.prevLink = this.getChapterPath(tutorial, higherChapters)
  } else if (!!tutorial.pathId && tutorial.higherItem) {
    let higherTutorial = tutorial.higherItem
    lastChapter = higherTutorial.chapters[higherTutorial.chapters.length-1]

    if (lastChapter.orderIndex == 1) {
      this.prevLink = this.getTutorialPath(higherTutorial)
    } else {
      this.prevLink = this.getChapterPath(higherTutorial, higherChapters)
    }
  }

  this.storeProps = {
    checkableType: 'Chapter',
    checkableId: chapter.id,
    checkboxes: this.signedIn ? currentUser.getCheckboxesFor(chapter) : []
  }

  this.setStore()

  this.modal = this.getModalChapter(chapter) || this.getModalTutorial(tutorial)
}
```

Parameters and Return

- A function had better be able to get some input and give output as well; otherwise, each function would be independent. Similar to functions in Math.

- Remember to give each parameter's value(argument) when calling a function.

General Syntax of Functions



Example

```
def add_numbers(num1, num2):
    sum = num1 + num2
    print("Sum: ", sum)
    return sum


result = add_numbers(5, 4)
```

Multiple Return Values

- Like being able to have multiple parameters, we can return multiple values.
  return x, y, z

- Remember to make the call accordingly.

- x, y, z = a_function_returning_three_values()

Benefits of Using Functions

- Structured code
- Abstraction
- Reusability of the same code again and again (Just change the arguments if needed)

Benefits of Using Functions (Cont.)

- Others can use your code as well.
- Or vice versa: Python libraries
- Easier to make changes and fix bugs

User Defined Functions vs Built-in Functions

- What we covered was user-defined functions.

- Python already provides some built-in functions like input() and print()

Thanks

Any questions?

References

1. http://www.quickmeme.com/p/3w1bz5/page/3
2. https://programmerhumor.io/wp-content/uploads/2023/10/programmerhumor-io-programming-memes-e315529c6afdef1.jpg
3. https://www.geeksforgeeks.org/python-functions/