PALs CSC-152

Section 02, Professor Kim, M-W-F

Tony Stone

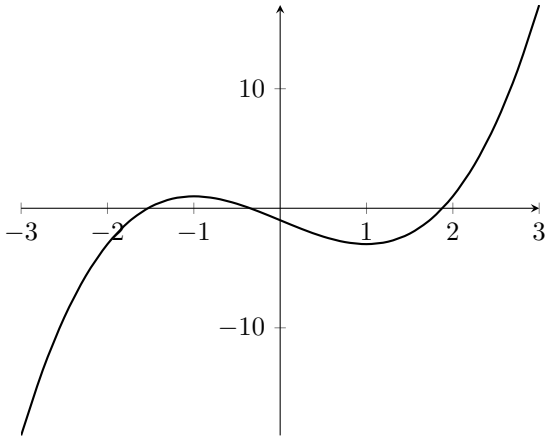
November 12, 2025

Lesson V

1 Functions

1.1 Abstract Consideration of Functions

What exactly do we mean by "function?" Let's first think of this in the general sense. In math, when we are referring to a function, we are talking about a thing that takes an input, and gives an output. Consider a function f, where $f(x) = x^3 - 3x - 1$ Let's plot this!



Here, we see that for a given input x, there is a corresponding output $f(x)$. In the very abstract sense, we have a defined way of doing something ($f(x)$) to get something else (x).

1.2 Writing Functions

When we're looking to write our own functions, this paradigm is an important idea. We will usually have some *thing*, and we want a way to do something to it. I am being intentionally broad here. There is not much to limit what a function can, or cannot be.

To root ourselves back into computer science, suppose we have a string literal, and we want to search it for a specific character. This can be coded very plainly for the specific string.

```
1 char = 'a'
2 for c in "Hello":
3     print("found") if char is c else print("not found")
4     # Note the formatting of this if-else statement!
5     # Python sometimes has different ways to phrase stetements
6     # Another one for the toolbox!
```

But this only works for "hello" and the character "a," are we really going to re-write a for loop every time we want to check a character? Of course not! We have something: in this case a string and a character. We'd like to do something to it: search said string for said character. This is practically asking to be a function. Let's see that.

```
def findCharacterBasic(astring, achar):
    for c in astring:
        print("found") if achar is c else print("not found")
```

Basically the same thing right? But actually, this will allow us to input any string, and any character! The use of the function has allowed us to completely define this search operation in terms of variables. To so, we simply **call the function**:

```
findCharacterBasic("sue", "s"):
```

Output:

```
found
not found
not found
```

The variables that get passed into the function are known as a functions **Arguments**. The function we have written is known as a **void** function. A void function is a function that upon being called, that's it, we just call them! In our case, the function printed something in the console, but the function didn't give anything back to us. You can think of this as a one-way interaction, we call the function, and it goes off and does it.

Rarely do we want to print something out from within a function. Interactions with the console is the job of the main portion of our code. Console interactions are more-or-less hard coded, the beauty of functions is their flexibility, so don't bind them with console output.

1.3 Returning Functions

What we can do is write a function so that it will **return** a value (or many) so that we can store it, and do with it what we will. This is now a two-way interaction, where we call a function, and then it hands us something back.

A void function, or a function with a return statement is not inherently better or worse, and entirely use dependent. The world of **object oriented programming** (OOP) is made possible based on both of these function paradigms. We are not concerned with OOP right now, just keep this in the back of your mind!

Let's edit the above function to return something that we can store in a variable, and then make a decision about what to do with it.

```
def findCharacterEdited(astring, achar):
    found = False
    for c in astring:
        if achar is c:
            found = True
    return found

check = findCharacter("sue", "s")
if check is True:
    print("found")
else:
    print("not found")
```

Output:

```
found
```

Our for loop will run for every character in the string we provide it. We initialize a variable found to be false. As the loop iterates, if the character in the given iteration matches the character that we're checking for, it will assign it to be true, as we have found the character. Finally, our function returns the value of found, indicating the character's inclusion in the string.

When the return statement executes, this signifies the end of that function. This is particularly helpful with a conditional case; if something is true, return something, else, return something else! Both cases end the loop, but only return one thing. This will be much more helpful in later CS endeavors involving **recursion**. For now, just one return statement that returns values all properly organized within the function works just fine!

Let's backtrack a moment, a few times now I have mentioned functions returning several values. What does that look like?

```
def parseCharacter(astring, achar):
    found = False
    notChar = []
    for c in astring:
        if achar is c:
           found = True
        else:
            notChar.append(c)
    return found, notChar

checkIncluded, nots = parseCharacter("sue", "s")
print(checkIncluded, nots)
```

Output:

```
True ['u', 'e']
```

We are now keeping track of all of the characters that are not the character which we are parsing the string for. In the return statement, we can send this list back by separating variables with a comma. Notice how in the main portion where we are calling the function, we need to use two distinct variables in order to accept the functions output. We then print those two values, in no particularly fancy way.

2 Key Terms

- **Function**: A process that can be applied to types of items in a specified way.
- **Calling a function**: The act of stating the functions name to invoke it's action on an item.
- **Argument**: The variable that gets input into a function.
- **Void functions**: A function which does not return a value. 'void' or a word like 'void' is often used as a keyword in other languages.
- **return**: The keyword which allows functions to provide values.

3 Availability

- In Lecture:
  - Wednesday: 10:00am - 11:00am
- PAL Sessions:
  - Monday: 2:00pm - 3:00pm
  - Wednesday: 9:00am - 10:00am
  - Wednesday: 2:00pm - 3:00pm

References

[1] Tony Gaddis. *Starting Out with Python*. Pearson, 5th edition, 2021.

[2] GeeksforGeeks. Python Tutorial — Learn Python Programming Language - GeeksforGeeks — geeksforgeeks.org. https://www.geeksforgeeks.org/python/python-programming-language-tutorial/. [Accessed 29-10-2025].

[3] Bradley N. Miller and David L. Ranum. Problem solving with algorithms and data structures using python. https://runestone.academy/ns/books/published/pythonds/index.html, 2014. Interactive online edition — accessed: 2024.

[4] Python Software Foundation. *Python: A dynamic, open source programming language*, 2025. Version 3.x documentation.

[5] W3Schools. W3Schools.com — w3schools.com. https://www.w3schools.com/python/default.asp. [Accessed 29-10-2025].