Lecture 6: Parameterized Functions

CS 51P

September 23, 2019

Review: Defining Functions

- Why?
  - There's some useful operation that you want to do over and over and over
  - Easier to read/understand
  - Easier to modify/change/debug

- How?

header

body

return

```
def sum_squares():
  num = int(input("pos int?\n"))
  sum = 0
  for i in range(1, num+1):
    sum = sum + num**2
  return num
```

Review: Calling Functions

```
def sum_squares():
  num = int(input("pos int?\n"))
  sum = 0
  for i in range(1, num+1):
    sum = sum + num**2
  return num

sum = sum_squares()
print(sum)

# or

print(sum_squares())
```

Main functions

- By convention, the only code that goes in the body of a Python file is the two-line program

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

- The rest of the program is defined in a function called main()
- (or in other functions!)

```
def sum_squares():
  num = int(input("pos int?\n"))
  sum = 0
  for i in range(1, num+1):
    sum = sum + num**2
  return num


def main():
  sum = sum_squares()
  print(sum)


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

Boolean Return Values

- Functions can evaluate to a value of any type
- …So functions can be Boolean expressions
- …So functions can be conditions!

- We've actually seen this before
  - e.g., if str.isdigit(input_string):

Example

- Define a function called good_choice() that asks the user for a positive integer and evaluates to True if the user enters 13 and False if they enter anything else?

- We want to be able to use the function as follows:

```
def main():
  if good_choice():
    print("yay")
  else:
    print("boo")
```

Exercise

```
def mystery():
    x = input()
    i = 0
    m = 1
    n = 0
    for c in x:
        if i == 0 and c == '-':
            m = 2
        elif c == '.':
            n = n+1
        elif not str.isdigit(c):
            return False
        i = i + 1
    return i >= m and n <= 1
```

- What does the function mystery() do?
- What would be better names for the variables x, i, m, and n?

What if you wanted your good_choice function to be able to check for numbers other than 13?

Parameterized Functions

- Functions can be defined with **parameters**, special variables that can be used inside the function and that are defined when the function is called

- Defining a parameterized function:

```
def good_choice(n):
    x = int(input("pos int?\n"))
    return x == n
```

parameter

- Calling a parameterized function:

```
b = good_choice(13)
```

argument

Example

- Define a function called `square` that takes a number n (an `int` or `float`) as a parameter and returns that number squared

- Define a function called `sum_squares` that takes a number n (an `int`). If the number is a positive `int`, it returns the sum of the squares 1,…, n. Otherwise it returns 0.

Exercise

- Define a function is_pos_int that takes a string and returns True if the string represents an integer value and False otherwise

- Write a main function that uses the functions get_pos_int and sum_squares to get a positive integer from the user and then print the sum of the squares from 1 to that number

Multi-parameter Functions

- Define a function called `area` that takes two numbers l and w (an `int` or `float`) as parameters and returns the area of a rectangle with length l and width w

- Note: parameters can also be optional!

Docstrings

- "A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition."
- every file should start at the top with a multiline comment that gives the author, date, description of what the code does
- every function header should be followed by a multiline comment that describes what the function does, specifies any input parameters, and specifies the return type/value

```
def square(n):
    """
    Computes the square of n
    :param n (int or float): a number
    :return (int or float): n*n
    """
    return n * n
```