Intro to Programming

3-26-19

Sign In!

https://tinyurl.com/CADIntroPySp19-5

Last Week's Review

- Using a `for` loop and the range function, calculate the first 10 square numbers and store them into a list.
    - `squares` should read `[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]`
- Starting from the code we've written together, modify it so that squares now only stores odd values. If a square is even, print the base along with a message: `"The square of 2 is even!"`

Last Week's Review Answer

```python
squares = []
for value in range(1,11):
    square = value ** 2
    squares.append(square)
```

HINTS:

- Use an `if/else` statement
- Use the `%` operator (remainder division)

- EX: `5 % 5 = 0`
- (5 divides evenly into 5. There's no remainder, so the result is 0)
- EX: `5 % 2 = 1`
- (2 doesn't divide evenly into 5. As such, there is a remainder of one.)
- By doing `% 2` on any number we can check if it is odd or even
- **odd % 2 = 1, even % 2 = 0**

Last Week’s Review Answer

```
squares = []
for value in range(1,11):
    if value % 2 == 0:
        print(“The square of “ + value + “ is even!”)
    else:
        square = value ** 2
        squares.append(square)
```

While Loops

- `while` loops can repeat **while** the specified condition is `True`
- The syntax is very similar to a `for` loop or `if` statement
- Once the condition evaluates to `False`, we exit the loop and continue with the rest of the code

```
number = 1
while number < 5:
    print("number: " + number)
    number = number + 1
```

OUTPUT:

```
number: 1
number: 2
number: 3
number: 4
```

Breaks and Continues

Breaks:

- When you see this, exit whatever loop you're in immediately

```
number = 1
while True:
    if number == 3:
        break
    number += 1
print(number)
```

iteration	loop condition is `True`	number
1	Yes	1 → 2
2	Yes	2 → 3
3	Yes	3

```
OUTPUT: 3
```

Breaks and Continues

Continues:

- When you see this, skip the iteration you're in and go to the next iteration

```
number = 1
while number < 5:
    if number == 2:
        number *= 2
        continue
    number += 1
print(number)
```

iteration	loop condition is `True`	number
1	Yes	1 → 2
2	Yes	2 → 4
3	Yes	4 → 5
4	No	5

```
OUTPUT: 5
```

While Loop Exercises

What’s wrong with the following code? How can I fix it?

```
age = 16
while age < 30:
    if age == 16:
        print(“I can drive!”)
        continue
    if age == 18:
        print(“I can vote!”)
        continue
    age += 1
```

While Loop Exercises

What’s wrong with the following code?

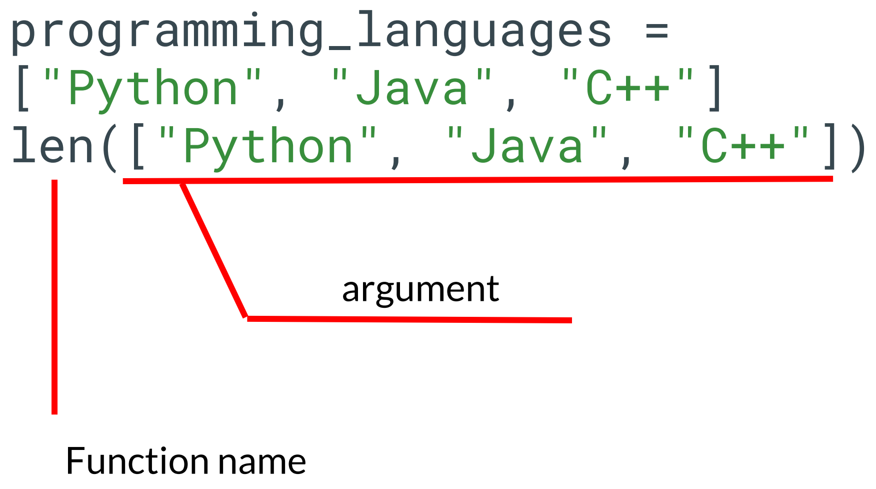
```
age = 16
while age < 30:
    if age == 16:
        print(“I can drive!”)
        continue
    if age == 18:
        print(“I can vote!”)
        continue
    age += 1
```

How can I fix it?

```
age = 16
while age < 30:
    age += 1
    if age == 16:
        print(“I can drive!”)
        continue
    if age == 18:
        print(“I can vote!”)
        continue
```

Functions

- Functions are blocks of code designed to do a specific task that you can re-use
- We have already used functions - len(), print(), int(), etc.
- Functions are called with their name and any optional arguments enclosed in parentheses



```
programming_languages =
["Python", "Java", "C++"]
len(["Python", "Java", "C++"])
```

Writing functions

- Function are defined using the def keyword
- The function name, given at ① must be followed by parentheses and a colon.
    - All other lines of the function must be indented one level
- It's good practice to write a docstring, which is a comment describing what the function does

```
def greet_user(): ①

    """ Display a simple greeting. """ ②

    print("Hello!") ③

greet_user() ④

Output:

Hello!
```

Passing information to a function

- Modified slightly our function greet_user() can greet people by name
- By adding the *parameter* `username` you allow the function to accept any value you specify
- The function will now expect us to provide a value for `username` each time you call it

```
def greet_user(username):
    """ Display a simple greeting. """
    print("Hello," + username.title + "!")

greet_user('claire')
greet_user('sean')
greet_user()

Output:
Hello, Claire!
Hello, Sean!
TypeError: greet_user() missing 1 required
positional argument: 'username'
```

Multiple arguments

```
def describe_pet(animal_type, pet_name):
    """ Display information about a pet. """
    print("I have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() +
".")

describe_pet("dog", "august")
print()
describe_pet("august", "dog")

Output:
I have a dog.
My dog's name is August.

I have a august.
My august's name is Dog.
```

- There are multiple ways to pass your function multiple arguments
- The default is with *positional arguments* which need to be in the same order the parameters were written
- Order matters! Otherwise errors can occur.

Return values

- Functions don’t have to display their output directly
- Instead they can process some data and return a value or set of values
- The value a function returns is aptly called the *return value*
- The return statement takes a value from inside the function and sends it back to the line that called it
- The code on the right results in no output. What’s wrong with it and how can we fix it?

```
def total_price(price, tax_rate):
    """
    Calculates the price after tax.

    price -- float
    tax_rate -- float, given as a decimal
    """
    tax = price * tax_rate
    tax = round(tax, 2)
    return price + tax

total_price(5, .0825)
```

Return values

- Functions don’t have to display their output directly
- Instead they can process some data and return a value or set of values
- The value a function returns is aptly called the *return value*
- The return statement takes a value from inside the function and sends it back to the line that called it

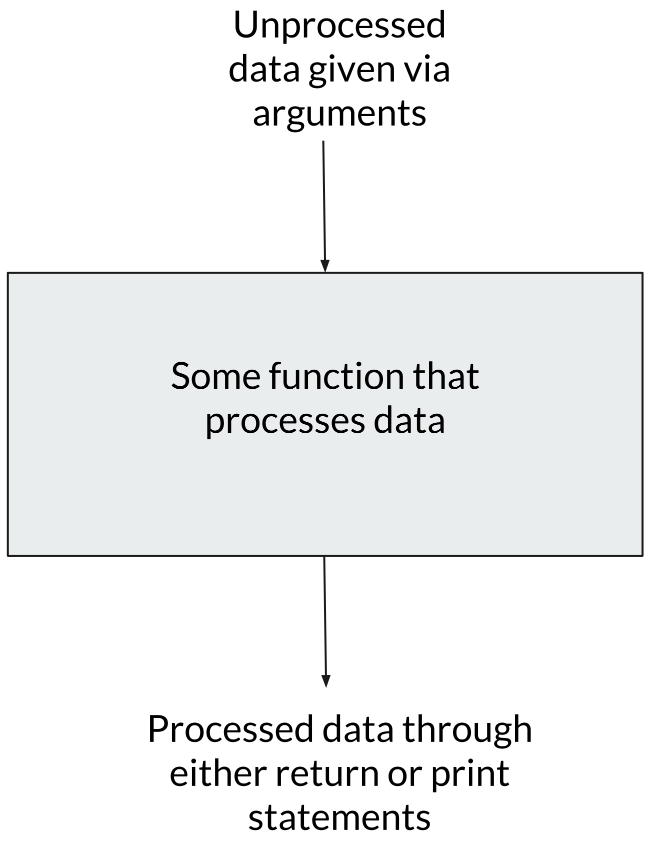
```
def total_price(price, tax_rate):
    """
    Calculates the price after tax.

    price -- float
    tax_rate -- float, given as a decimal
    """
    tax = price * tax_rate
    tax = round(tax, 2)
    return price + tax

print(total_price(5, .0825))
```

Output:

```
5.41
```

Function scope

- Think of functions as “black boxes”
- Variables defined inside the function cannot be accessed outside of the function
- Anything defined within the function is referred to as being within the function’s “scope”
- Sometimes variables can be accessed from outside a function, and sometimes it’ll throw an error. The reasons why are beyond the scope of this course.
    - To be safe, always pass functions data through their parameters.


Unprocessed data given via arguments
Some function that processes data
Processed data through either return or print statements

https://goo.gl/D3N7xf

Function exercise

You are driving a little too fast, and a police officer stops you. Write code to compute the result, encoded as an int value: 0=no ticket, 1=small ticket, 2=big ticket. If speed is 60 or less, the result is 0. If speed is between 61 and 80 inclusive, the result is 1. If speed is 81 or more, the result is 2. Unless it is your birthday -- on that day, your speed can be 5 higher in all cases.

```
caught_speeding(60, False) → 0
caught_speeding(65, False) → 1
caught_speeding(65, True) → 0
```

```
def caught_speeding(speed,
is_birthday):
  return None
```

Using functions we've written

```
my_speed = 75
my_birthday = False

print("I am going " + my_speed + "mph.")
ticket = caught_speeding(my_speed, my_birthday)
if ticket:
    print("I was caught speeding.")
    print("My ticket was $" + 100 * ticket)
else:
    print("I was not caught speeding.")
```

Thanks for coming!

- Next week - TBD
- Please fill out our feedback form, especially if you'd like to specify which topics we cover next week!
    - https://tinyurl.com/CADIntroPyFeedback5