Week 2: Functions
FINALLY!
Remember when I bragged that Python has lots of built in tools and libraries? Some built in functions:

- `print()`
- `type()`
- `help()` **doesn’t need a print, take that consistency! More on this later!**
- `min()` / `max()`
- `bin()` / `hex()` / `oct()`
- `id()`
- `input()`
- `int()` / `float()`
- `pow()` `round()`
More examples of built-in functions

Just to name a few…

<table>
<thead>
<tr>
<th>abs()</th>
<th>bytearray()</th>
<th>enumerate()</th>
</tr>
</thead>
<tbody>
<tr>
<td>dict() help()</td>
<td>filter()</td>
<td>input()</td>
</tr>
<tr>
<td>min()</td>
<td>issubclass()</td>
<td>oct()</td>
</tr>
<tr>
<td>setattr()</td>
<td>pow()</td>
<td>bin()</td>
</tr>
<tr>
<td>all()</td>
<td>super()</td>
<td>eval()</td>
</tr>
<tr>
<td>dir()</td>
<td>bytes()</td>
<td>int()</td>
</tr>
<tr>
<td>hex()</td>
<td>float()</td>
<td>open()</td>
</tr>
<tr>
<td>next()</td>
<td>iter()</td>
<td>str()</td>
</tr>
<tr>
<td>slice()</td>
<td>print()</td>
<td>bool()</td>
</tr>
<tr>
<td>any()</td>
<td>tuple()</td>
<td>exec()</td>
</tr>
<tr>
<td>divmod()</td>
<td>callable()</td>
<td>isinstance()</td>
</tr>
<tr>
<td>id()</td>
<td>format()</td>
<td>ord()</td>
</tr>
<tr>
<td>object()</td>
<td>len()</td>
<td>sum()</td>
</tr>
<tr>
<td>sorted()</td>
<td>property()</td>
<td></td>
</tr>
<tr>
<td>ascii()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arguments

- Not that kind of argument

- An argument is something passed to a function, it’s what you want the function to work on. Functions can be thought of as black boxes

- Aka a parameter.
Why Use functions?

“Off the top of my head, I'd say you're looking at a Bowski, a Jim Brown, a Miss Daisy, two Jethros and a Leon Spinks, not to mention the biggest Ella Fitzgerald ever!”

Or for those of us who are normal: Reusability.

And unlike the previous example: Clarity.
Some useful built-in functions:

1. help()
min() / max()

- Running from Math? Python can help!
int/float/str

- Casting as we discussed earlier
print() / input()

- The basic input and output functions in python
Quick Question:

- If we had to accept 2 numbers from a user, and check which one of the two was greater, how would we do that?

- ...and one last one. Take two numbers from a user and add them.
BUT WAIT,

THERE'S MORE!
Since we will not be having class on Friday, We will have the test on Thursday during class.

There is a Project this week, I will assign it on Friday. Please check blackboard for it.
WELCOME BACK!
## Where we are:

<table>
<thead>
<tr>
<th>Types</th>
<th>Functions</th>
<th>Flow Control</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>print()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Float</td>
<td>input()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>pow()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolean</td>
<td>int()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>float()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>str()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>min()/max()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>help()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Built-In Functions

- Most of us like just the regular chocolate-chip or peanut butter or snickerdoodle cookie varieties.
- But what if I (or the Dalai Lama) wanted one with everything?

- Similarly, If we have a whole lot of built in “flavors” (read: functions) in python. But what if we wanted our own flavor?
User Defined Functions
def times_two(num):
return num * 2

- `def` is a key word that tells Python you are starting the
definition of a function
- `times_two` is the name of my function
- `num` is a parameter (or argument), it is an input passed
to the function, not all functions require arguments
- `return` is what the function is going to give back when
finished

Let's try this code, do you think it will work?
User Defined Functions: Indent

- Why didn’t that code work?
  - Because we forgot a crucial part of function writing! The indent
  - Try the one given below.

```python
def times_two(num):
    return num * 2
```

- Luckily for us, IDLE does this automatically when it sees the keyword `def` and the `:`.
- For the most part, python is flexible with whitespaces, the biggest exception to this is the indent.
Indent continued:

- So why did it work?

```python
def times_two(num):
    return num * 2
```

- Python uses indents to tell what code goes together
- when the code stops being indented then python knows the function is complete
- so

```python
def times_two(num):
    return num * 2
```
- won’t work because the function times_two has no code
The “other” argument

```python
def times_two(num):
    return num * 2
```

- `num` is a parameter (or argument), it is an input passed to the function, not all functions require arguments.
- What exactly is “num”? It’s essentially a variable, but one that only lives inside the function.
- If we call `times_two(4)` then the first thing this code does is `num = 4`.
- Arguments are what let us call functions on a variety of inputs.
A Special kind of User-Defined Function: The Hard Coded Function

def three_times_two():
    return 3 * 2

- We’ve written a version of times_two that doesn’t take an argument and instead is hardcoded for a specific value (i.e. fixed, not variable).
- this works the same way as times_two(3) would but is obviously much less useful.
def times_two(num):
    return num * 2

def times_two(num):
    print(num * 2)

- Do these do the same thing? Hint: **NO**.
- Note the color differences,
  orange is a keyword,
  purple is a built in function
- What does the second function *return*?
So what does it all mean?

- 42

- Just Kidding. Simply put:
  - `print()` exists to give information to a human being
  - `returns` exist to pass data around between parts of the program

- Let's take the examples of
  
  ```python
  x = max(2, 3)
  ```

- And,
  
  ```python
  print(max(2, 3))
  ```
50 shades of IDLE

Ok there aren’t so many but here are the ones that are there

Python default syntax colors:

- **Keywords**: orange
- **Builtins**: royal purple
- **Strings**: green
- **Comments**: red
- **Definitions**: blue

Shell default colors:

- **Console output**: brown
- **stdout**: blue
- **stderr**: red
- **stdin**: black

This is also viewable on IDLE Help on the taskbar
Programming as Data

- a function is essentially a variable whose “value” is a series of steps on some input. This was a HUGE conceptual breakthrough.
WELCOME BACK!
### Where we are:

<table>
<thead>
<tr>
<th>Types</th>
<th>Functions</th>
<th>Flow Control</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>print()</td>
<td></td>
<td>def</td>
</tr>
<tr>
<td>Float</td>
<td>input()</td>
<td></td>
<td>return</td>
</tr>
<tr>
<td>String</td>
<td>pow()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolean</td>
<td>int()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>float()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>str()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>min()/max()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>help()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Verbosity!

This code

```python
def foo(a):
    return a * a
```

is a lot less easy to understand than this

```python
def square(num):
    return num * num
```

Just like with variables giving functions and arguments good names is a very good idea (which makes sense since arguments and functions really are sorts of variables)
def foo(A):
    doubleA = 2*A

print(doubleA)

Does this code work?
BUGS!!

SHIT

HAPPENS
A Quick Introduction to Entymology

- Syntax Errors
- Logic Errors
- Runtime Errors

This list is in ascending order of suck.

This list is non-exhaustive, there are many more types of errors and all come under the category of exceptions.
def times_two(num: 
    return num * 2
max(2, 3)

def two()
return 2

Syntax error = your code sucks (or a typo)

Good news- easy to catch, easy to fix
Logical Errors

```python
def times_two(num):
    return num * 3
```

- Logic error = your *computational thinking* sucks (or a typo)

- May be easy or hard to spot, often frustrating to fix
“good” runtime error:

```python
def times_two(nam):
    return num * 2
```

bad runtime error

```python
def divide_ten(num):
    return 10 / num
```

- Runtime error = you didn’t think of an important case, or you referenced non existing variables

- Can be nearly impossible to find without very good test cases. Often not that hard to fix.
For more on errors and exceptions

- https://docs.python.org/3.4/library/exceptions.html
Group Question:
Given the information that simple interest is calculated with the formula

\[ S.I = \text{Principle Amount} \times \left( \frac{\text{Rate}}{100} \right) \times \text{Time (in years)} \]

Write a function to calculate Simple interest