Day 3

Defining Functions

You can define your own functions using `def`.

This is important to avoid repeating yourself.

Functions have a **name**, **parameters**, and **return** a value (usually).

Defining Functions

```
def is_big(number):
    if number > 100:
        return True
    else:
        return False

print(is_big(10))
print(is_big(200))
```

Number-Finder Take Two

Our previous program for finding the smallest number whose square was bigger than a certain other number only worked for a **hard-coded** number.

If we wanted to find n for multiple square values, we would need to copy the code or write a new algorithm.

Defining a **function** lets us reuse our algorithm easily.

Number-Finder Function

Our previous code, for reference:

```
still_searching = True
current_number = 0
while still_searching:
  square = current_number * current_number
  if square > 2000000:
    print(current_number)
    print(square)
    still_searching = False
```

```
  else:
    current_number = current_number + 1
```

Number-Finder Function

```
function find_smallest_square_bigger_than(target_number):
  current_number = 0
  while True:
    square = current_number * current_number
    if square > target_number:
      return current_number
    else:
      current_number = current_number + 1
```

Note, we no longer need still_searching, as the loop ends when we **return**.

Lists

Data structures are used to represent data that is related in some way.

The **list** data structure is one of the most common and useful ones you will encounter.

A list is a number of **elements** in a specific order.

Creating Lists

Use square brackets to create a list:

```
empty_list = []
my_awesome_numbers = [35, 128, 2]
words = ["cat", "dog", "elephant"]
color = "blue"
colors = [color, "red"]
```

Getting Elements from a List

You can access an element at an **index** using square brackets:

```
animals = ["cat", "dog", "elephant"]
animals[0]    # "cat"
animals[1]    # "dog"
animals[2]    # "elephant"
animals[3]    # IndexError: list index out of
range
```

Adding to Lists

You can add elements to the end of a list using `append`:

```
animals = ["cat", "dog", "elephant"]
animals.append("bird")
animals[3]   # "bird"
```

Removing from Lists

Removing elements is done with `remove`:

```
animals = ["cat", "dog", "elephant"]
animals.remove("dog")
animals[1]   # "elephant"
```

Getting List Size

Call the `len` function on a list:

```
animals = ["cat", "dog", "elephant"]
len(animals)    # 3
animals.append("bird")
len(animals)    # 4
animals.remove("dog")
len(animals)    # 3
```

Iterating over Lists

It's very common to want to run a bit of code on every element of a list. You can do this using `for..in`:

```
animals = ["cat", "dog", "elephant"]
for animal in animals:
  print(animal + "!")
```

Algorithm: Selection Sort

It's common to want to sort a list in a particular order:

- Sort emails by time
- Sort scores high to low in a game
- ?

How might you sort a deck of cards?

A simple way to do this is the **selection sort** algorithm.

Selection Sort

Given a list to sort called `input`,

1. Create an empty list called `output`
2. Find the smallest element in `input`
3. Remove the element from `input`
4. Add the element to the end of `output`
5. If `input` is empty, return `output`
6. If `input` is not empty, go back to step 2

Selection Sort

```
def selection_sort(input):
  output = []
  while len(input) > 0:
    smallest = get_smallest(input)
    input.remove(smallest)
    output.append(smallest)
  return output
```

`get_smallest` Function

A key part of programming is breaking down big problems into smaller problems, then combining the solutions.

Can you think of an algorithm for `get_smallest`?

get_smallest function

```
def get_smallest(input):
  smallest = input[0]
  for element in input:
      if element < smallest:
        smallest = element
  return smallest
```

Break

Lab 3: Where's the Treasure?