Defining Functions

Today's plan

- Hand back lab 2
- Review Monday
  - Comparison of `for` and `while` loops
- Defining our own functions!

Review

- Python has a **random** library that we can import as follows:

```
from random import *
```

- There are several functions in the random library. We've seen `randrange(start, stop, step)`.

```
# Returns 0, 1, or 2, chosen at random
>>> randrange(3)
0
>>> randrange(3)
2
```

Review

- A `while` loop has a boolean expression and an indented block of code. We repeatedly perform the indented code until the boolean expression evaluates to `False` for the first time.

- If the boolean expression in a `while` loop always evaluates to `True`, this is an **infinite loop**.

- To kill a program in an infinite loop, we type Ctrl-c in unix

`for` vs. `while`

- Often when programming, you intuit that some task needs to be performed repeatedly within a program. This calls for a loop.

- A general rule of thumb:

  - If your program can calculate how many times the task needs to be repeated, use a **`for`** loop. Often this happens when we want to do some set of tasks for each item in a sequence.

  - If it can't, use a **`while`** loop. Figure out a boolean expression that will evaluate to **`True`** as long as the task should be repeated, **`False`** as soon as the task should stop repeating.

`for item` vs. `for i`

- If you've decided to go with a `for` loop, which kind?

- All `for` loops iterate over a sequence, but sometimes that sequence should contain the items we want to use (`for item in L:`) other times it should contain the indices of the items we want to use (`for i in range(len(L)):`)

- `for char in S:` is like `for item in L:`

for item/char

```
text = raw_input("Enter text: ")
doubled = ""
for char in text:
    doubled = doubled + char*2
print(doubled)
```

```
for char in text:
    doubled = doubled + char*2
```

equivalent, less convenient, more general:

```
for i in range(len(text)):
    char = text[i]
    doubled = doubled + char*2
```

for i

```
for i in range(len(text)):
    print(text[:i] + "a" + text[i+1:])
```

equivalent, but less convenient:

```
i = 0
for char in text:
    print(text[:i] + "a" + text[i+1:])
    i = i + 1
```

```
for i in range(len(text)):
    print(text[:i] + "a" + text[i+1:])
```

equivalent, less convenient, more general:

```
i = 0
while i < len(text):
    print(text[:i] + "a" + text[i+1:])
    i += 1
```

Requires a `while`

```
secret_number = randrange(1, 101)
guess = int(raw_input("Guess number: "))
while guess != secret_number:
    if guess > secret_number:
        print("Too high")
    else:
        print("Too low")
    guess = int(raw_input("Guess again: "))
print("You got it!")
```

How functions work

- When we **define** a function we list its **parameters**

- When we **invoke** or **call** a function we list its **arguments**. If the function ends with a **return** statement, the invocation evaluates to the value returned.

- The arguments determine the initial values of the parameters, based on order.

- Each function has its own set of variables, distinct from the variables defined within other functions.

- A function may have side effects, like printing or gathering user input.

Defining functions

Functions are useful

- Why use functions?
  - Keep our programs organized, easy to follow
  - For planning / outlining
  - Easier to maintain
  - Easier to reuse
  - Separation of concerns