CS5001
Spring 2020
Handout: Input + Output

All programming languages must be able to:

1. remember things,
2. repeat things,
3. communicate, and
4. make decisions.

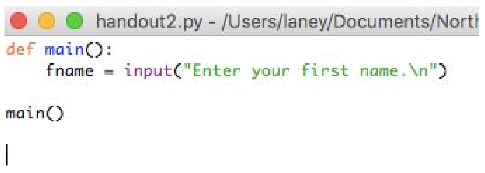
Input/output is all about #3: communicating. *Input* refers to any way that we collect information we need from another source --information we use in our program could come from the user, a file, or another computer. *Output* refers to any way that we report something we've computed to another source -- information we report could go to the user, a file, or another computer.

For now, let's focus on communicating with the user. We all have lots of experience being users of compute programs. We're users of web browsers, word processors, Instagram, Facebook, Kwazy Kupcakes, FriendFace, etc. When we use one of these programs, we don't see or care about the source code, we just interact with the program.

Whenever we write a Python program, we have that user in mind -- someone who doesn't see or care about our source code but who uses our program because it's useful and/or fun.

Input

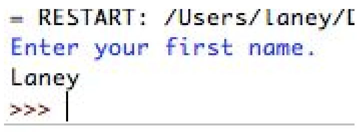
In Python, user input comes when they type something in the terminal in response to our prompt. Python has a built-in function called **`input`**. With this function, we specify the prompt (a string), and Python will wait patiently on the terminal until the user types something and hits enter.



```
handout2.py - /Users/laney/Documents/Nortl
def main():
    fname = input("Enter your first name.\n")

main()
```

Source code we write to prompt the user.



```
= RESTART: /Users/laney/D
Enter your first name.
Laney
>>> 
```

User sees the prompt and types in the terminal (they don't care about the source code).

The **`input`** function always gives us a string. The variable **`fname`** will store a string in this example. Having a string by default is really common in input/output.

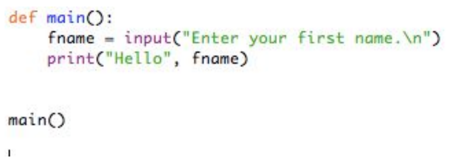
If you want to prompt the user for an integer, you need to do a little conversion, which looks like this:

```
age = int(input("What is your age?\n"))
```

We still have input in there, we just also tell python we need to turn the usual string into an int.

Output
In Python, user output comes when we report something on the terminal -- usually the result of a calculation, or if we've gathered some information we want the user to see.

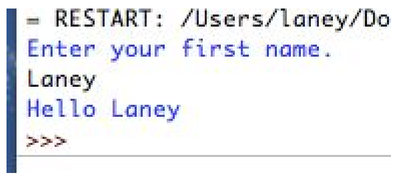
Python has a built-in function called **`print`**. We can give both literals and variables to this function and it puts them all together into one thing on the terminal.



```
def main():
    fname = input("Enter your first name.\n")
    print("Hello", fname)


main()
```

Source code we write to print to the terminal.



```
= RESTART: /Users/laney/Do
Enter your first name.
Laney
Hello Laney
>>>
```

User sees the output in the terminal (they don't care about the source code).

Unlike the **`input`** function, the **`print`** function can handle literals and variables of pretty much any data type. We just put commas in between them, like we've done above with the literal **`"Hello"`** and the variable **`fname`**.

You can mix-and-match as many variables and literals in the print function as you like. Python automatically puts a space after each one, and a linebreak at the end. Here's one that uses both fname (a string) and age (an int):

```
fname = input("Enter your first name.\n")
age = int(input("Enter your age.\n"))
print("Hello", fname, "I see you are", age, "years old."))
```

There are a lot of ways you can tweak how the output looks in terms of spacing, linebreaks, and other formatting. We'll just throw one out there for now, which is **`format`**. This is a super useful function for, among other things, specifying how many digits to print after a decimal point.

For example, here's how we would print my age as a float with exactly one digit after the decimal point:

```
print("Hello", fname, "I see you are {:.1f}".format(age), "years old.")
```

It looks super tricky, but here are the only key parts you need to know about

- `{ }` --- placeholder. Tells Python that some value will go here after calling format.
- `:.1` --- formatting type. Tells python to print one digit after the decimal
- `f` --- data type. Tells Python to treat the value as a float.
- `format(age)` --- calling the format function. Tells Python the thing being formatted is the variable age.

This is only one example, and there's a ton you can do to tweak print function. If you're ever stuck, use the help function in Python interactive mode. Try `help(format)` or `help(print)` -- they're useful!