WEEK 5:
FUNCTIONS

Procedures

This week we looked at subroutines, functions, procedures and scope. The first thing we looked at were procedures, which are a way of "bundling up" lines of code, giving them a name, and running them when we need them. Procedures allow us to reduce the need to repeatedly copy + paste blocks of code which are repeated throughout your source code. Instead, we can bundle them up together, and just invoke them where necessary!

For example, imagine you have written a piece of software which gives the user a text-based menu where they can select an action. The lines of code need to show this menu is something like 5-10 lines long, as the menu is this many lines too. Now, imagine you have to show this menu at a bunch of points throughout your program. What if you wanted to change the name of one action? You'd have to replace all instances of the name, one-by-one. This is why procedures as so handy!

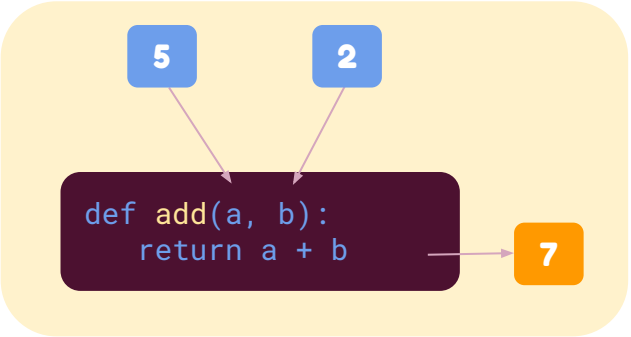
```
name = input("enter your name: ")
score = 0

if name != "your name":
   score = (len(name) ^ ord(name[0])) % 10
else:
   print_easter_egg();
   score = -500

print("Your name score is " + str(score));
```

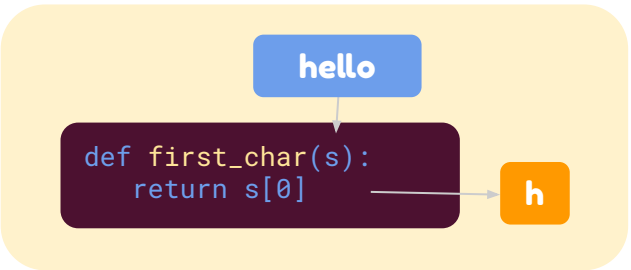
```
def print_easter_egg():
    print("haha. very funny.");
    print("for that, you get ");
    print("-500 points. ");
```

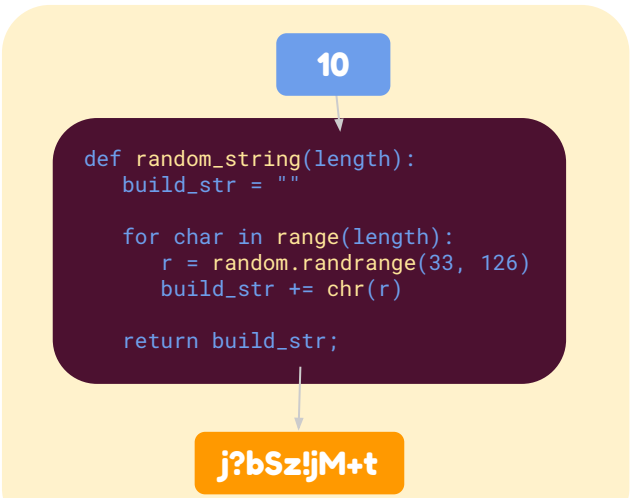
```
enter your name: your name

haha. very funny.
for that, you get
-500 points.

Your name score is -500
```

Experiment with procedures in Python:

1. Create a procedure which prints a poem of your choice. When the user types in the word "poem", call the procedure and show the poem. Remember:
   a. When defining procedures, you need to use the **def** keyword.
   b. Procedures need a name, followed by brackets and a colon.
   c. When invoking a (parameterless) procedure, you need to use two brackets to call it, like: **show_poem()**
2. Create a procedure which shows a custom message, and asks the user for input, using the **input()** function.
3. Create a procedure which uses a for loop to print the numbers 1-10 backwards, followed by the word "blast off!", and test that it works.







Functions & Parameters

We also looked at functions (procedures which "return" a value) and parameters. Functions are very useful tools -- they allow us to communicate and get answers from procedures. With a function, a value is "returned" from the function with the return key. This value can then be used in another portion of the code.

Another interesting feature are parameters. Parameters allow us to pass data into a function. We can then use that data in whatever calculation we perform. Where the "return" keyword passes data OUT of a function, parameters allow us to pass data INTO a function. This is a very powerful feature! Using these two tools (returning and parameters) we can build very useful functions which take some form of data and transform into another form.

For example, consider an "add" function (see the example on the left). This would take two numbers (lets call them a and b) and add them together. It would then "return" the result of this calculation: a + b.

Experiment with functions & parameters in Python:

1. Create a function called **`pi`**, which takes no parameters, and will always return 3.141592. Try print out the value of calling this function, by using **`print(pi())`**. What do you think you'll see?
2. Create a function called **`greet`**, which takes one parameter (the name of the person) and return a greeting with their name in it. For example, "Hello {name}!".
3. Using the random module (at the top of your code, put: import random), create a function called **`flip_coin`**, with no parameters. This should return "heads" 50% of the time, and "tails" 50% of the time:
    a. Can you expand this to take a parameter called **`heads_chance`**, which alters the probability you get a heads?

NEXT WEEK:

ARRAYS, LISTS & FILES

THE Ben AND Chris SHOW