CS 120: Intro to Computer Programming II

In-Class Activity - 05 Recursion - Day 3

This activity is all about **making predictions and testing them.** So there won't be code or pictures to turn in; instead, turn in a discussion of what your group thought, and what you learned. What were your predictions? What was the truth?

Activity 1 - Turn in this one

Consider the following function. **Without actually typing the code into an editor,** have everybody see if they can figure out, by hand, what will get printed out if we call `foo(4)`.

If you are one of the people who often talks in a group, **please leave space for others;** let's encourage the people who speak rarely to realy speak up here. We always want to do this, but with this ICA, we're going to be asking a whole series of questions which are **very easy to get mixed up.**

In this situation, it's imporant to allow people to **make mistakes** - they are expected! But let's use every mistake as a **learning opportunity.**

```
def foo(n):
    print(f"foo START: n={n}")

    if n == 0:
        print("  -- base case, doing nothing --")
    else:
        foo(n-1)

    print(f"foo END: n={n}")

foo(4)
```

Once the group has discussed their predictions, have somebody type the code into their favorite editor, and run it. Were you predictions correct? **Take time to discuss this as a group.** Probably, some of you made a mistaken prediction; let's see if we can understand why the code works the way it does!

Solution: This will print the various START messages, counting down from 4 to 0. It then prints the "base case" message, and then counts **up** with the END messages.

The activity continues on the next page.

Activity 2 - Turn in this one

Same instructions as before.

```
def mario(value):
    print(f"mario START: my value is: {value}")
    bowser(value)
    print(f"mario END: Oh no!  my value has been changed to: {value}")

def bowser(value):
    print(f"bowser START: I will steal half of the value!  old value: {value}")
    mine   = value//2
    value -= mine
    print(f"bowser END: mine: {mine}   remaining value: {value}")

mario(10)
print()
mario(-32)
print()
mario(15)
```

Solution: In this case, the `data` variable inside `mario` is **not** changed; this is because `bowser` is operating on a copy of the `value` variable; it is **not** the same thing!

Activity 3 - Turn in this one

Same instructions as before.

```
def luigi(data):
    print(f"luigi: START: my data is: {data}")
    waluigi(data)
    print( "luigi: END: No worries!  I know that other functions can't")
    print(f"            change my variables!  {data}")

def waluigi(data):
    print(f"waluigi: START: I will steal all of the data!  data: {data}")
    mine = 0
    for i in range(len(data)):
        mine += data[i]
        data[i] = 0
    print(f"waluigi: END: mine: {mine}   remaining data: {data}")

luigi([1,2,3])
print()
luigi([13,-26,12])
```

Solution: In this case, the array held by `luigi` **is** modified - because it is an array, the variable `data` inside `waluigi` is an **alias** of the array.

The activity continues on the next page.

Activity 4 - Optional

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

This recursive function is **broken** - but it's broken in a way that I've **often** seen students write their code. Just like last time, make a prediction about what the code will return. Once the group has made their predictions (and discussed it a bit) and run the code for real; see if we can help **everybody** understand why it returns what it does.

```
def broken_sum(data):
    count = 0

    if len(data) == 0:
        print("Array is empty, we're done!")
    else:
        count += data[0]
        broken_sum(data[1:])    # update the counter with the rest

    return count
```

Solution: This code will always return the value of the first element in the array (ignoring the rest of the array). The reason is because we do not add the return value into `count`.

Challenge Activity - Do not turn in this one

Make your predictions, and test this code as well:

```
def peach(data):
    print(f"peach: START: my data is: {data}")
    koopa(data)
    print( "peach: END: No worries!  I know that other functions can't")
    print(f"            change my variables!  {data}")

def koopa(data):
    print(f"koopa: START: I will steal all of the data!  data: {data}")
    mine = sum(data)
    data = [0,0,0]
    print(f"koopa: END: mine: {mine}   remaining data: {data}")

peach([1,2,3])
print()
peach([13,-26,12])
```

Solution: In this case, `peach` is safe. The difference, compared to the `luigi` example above, is that in this case `koopa` changes the `data` variable to a **new array**. Thus, it doesn't effect the `data` in `peach`!