CL07 – f-Strings & Intro to Recursion

Reminders

- **Quiz 00:** Regrade requests will be open **till 11:59pm Wednesday night!**
  - Please submit a regrade request if you believe your quiz was not graded correctly according to the rubric
- **LS07** and **CQ01** due tonight at 11:59pm

Want extra support? We're here and *want* to help!

f-strings (formatted string literals)

A helpful way to embed expressions directly into strings!

Without f-strings:

```
print("They are " + str(30 + 1))
```

With f-strings:

```
print(f"They are {30 + 1}")
```

Both will output the string:

```
They are 31
```

f-strings (formatted string literals)

```python
def get_class(subject: str, num: int) -> None:
    print(
        "I'm currently in "
        + subject
        + str(num)
        + ", but next semester I'm taking "
        + subject
        + str(num + 100)
        + "!"
    )


get_class(subject="COMP", num=110)
```

Will these two versions of the get_class function print the exact same phrase?

```python
def get_class(subject: str, num: int) -> None:
    print(f"I'm currently in {subject}{num}, but next semester I'm taking {subject}{num+100}!")


get_class(subject="COMP", num=110)
```

```
1  """Examples of conditionals."""
2
3
4  def number_report(x: int) -> None:
5      """Print some numerical properties of x"""
6      if x % 2 == 0:
7          print("Even")
8      else:
9          print("Odd")
10
11     if x % 3 == 0:
12         print("Divisible by 3")
13
14     if x == 0:
15         print("Zero")
16     else:
17         if x > 0:
18             print("Positive")
19         else:
20             print("Negative")
21
22     print("x is " + str(x))
23
24
25 number_report(x=110)
```

How could we convert the print statement on line 22 to use an f-string?

Your job: Diagram *at least* 2 function call frames…

But stop when you get tired or run out of lead!

```
1    def icarus(x: int) -> int:
2        """Unbound aspirations!"""
3        print(f"Height: {x}")
4        return icarus(x=x + 1)
5
6
7    print(icarus(x=0))
```

Questions to discuss with your neighbor(s):

What seems *problematic* about this function?

How might you prevent it?

```python
def icarus(x: int) -> int:
    """Unbound aspirations!"""
    print(f"Height: {x}")
    return icarus(x=x + 1)


print(icarus(x=0))
```

Stack Overflow and Recursion Errors

When a programmer writes a function that calls itself indefinitely (*infinitely*), the **function call stack** will *overflow…*

This leads to a **`Stack Overflow`** or **`Recursion Error`**:

```
RecursionError: maximum recursion depth exceeded while
calling a Python object
```

Base Cases and Recursive Cases

The key to writing recursive functions that are non-infinite!

To avoid StackOverflow Errors and infinite recursion:

1. You must have at least one **base case**
    a. Base case: a branch in a recursively defined function that **does not recur**
2. **Recursive cases** must change the arguments of recursive calls such that they make progress toward a base case

Trace the following program in a diagram:

```python
def icarus(x: int) -> int:
    """Unbound aspirations!"""
    print(f"Height: {x}")
    return icarus(x=x + 1)

def safe_icarus(x: int) -> int:
    """Bound aspirations!"""
    if x >= 2:
        return 1
    else:
        return 1 + safe_icarus(x=x + 1)

print(safe_icarus(x=0))
```

Checklist for developing a recursive function:

Base case:

- ❏ Does the function have a clear base case?
  - ❏ Ensure the base case returns a result directly (without calling the function again).
- ❏ Will the base case *always* be reached?

Recursive case:

- ❏ Ensure the function moves closer to the base case with each recursive call.
- ❏ Combine returned results from recursive calls where necessary.
- ❏ Test the function with edge cases (e.g., empty inputs, smallest and largest valid inputs, etc.). Does the function account for these cases?