Control Flow Practice

Tip: Calling Function Definitions in REPL

- When you are authoring a function and want to play around with calls to it you currently have three options available:

1. Define in a Python REPL
   - Downside: redefining the function each time you need to change it.

2. Define in a .py file and print out sample calls to it after its definition
   - Upside: easy to modify and add additional calls
   - Downside: if you print the return values of lots of calls it's painful to match up which calls led to which return values

3. New: Start a REPL that begins by evaluating the contents of a .py file
   - Upside: easy to modify the definition and try calling it!
   - Downside: must remember to quit the REPL and restart after changes to definitions.

- The command for #3 is: **`python -i directory/to/module.py`**
  - The REPL will now have all globally defined names (such as functions!) available for you to try calling them.

```python
import random


def main() -> None:
    x: int = ____
    y: bool = ____
    z: str = ____
    print(x, y, z)


def a() -> bool:
    return random.randint(0, 1) == 0


def b(whole_number: str) -> int:
    return int(whole_number)


def c(name: str, value: int) -> str:
    return name + ": " + str(value)


main()
```

Q1: There are three variable initialization statements. In each blank (three subsequent pollev questions), write a valid function call to an appropriate function defined below. You may use any literal values you'd like as arguments.

Environment Diagrams (v1)

1. Add columns for Call Stack, Heap, and Output
2. Add a Globals frame to Call Stack

Function Call

1. Verify and prepare for call
   i. Is function name bound in your diagram or built-in?
   ii. Fully evaluate each argument's expression
   iii. Do arguments match function parameters?
2. Establish new frame on call stack
   i. Add name of function
   ii. Add RA (Return Address line #)
   iii. Copy arguments to parameters bound in frame
3. Jump to first line of function definition

Function Return Statement

1. Evaluate returned expression
   - Add RV (Return Value) in current stack frame
2. Jump back to function caller
   i. Line is in RA (Return Address)
   ii. The function call evaluates to last frame's RV

Function Definitions: Enter name in current frame and draw arrow to Function object on heap labeled `Fn: [start_line] - [end_line]`

Current Frame: The most recently added frame that has not returned. (*No RV!*)

Name Resolution: Look for name in the current frame. Not there? Check Globals frame!

Variable Initialization: Enter name and space for variable in current frame.

Variable Assignment: Find variable's location via name resolution, copy assigned value to it.

Variable Access: Find variable via name resolution, use value currently assigned to it.

Q2 - Diagram the following code listing.

```
01 def main() -> None:
02     x: int = 1
03     x += f(x + 1)
04     print("x: " + str(x))
05
06
07 def f(x: int) -> int:
08     x += 2
09     return x
10
11
12 main()
```