Welcome!

We'll get started shortly ...

CS 49 Section

Week 8

Surajit A Bose

Agenda

- Logistics and check-ins
- Review of lecture concepts
  - Function inputs: parameters and arguments
  - Function output: `return` statements and values
  - Functions as black boxes
- Revised coding template
- Section Problems:
  - In Range (starter code here)
  - Medical Test Simulator (starter code here)

Logistics

How to get hold of me / get help+

- The section forum, 24 hr turnaround
- Email: bosesurajit@fhda.edu, 24 hr turnaround
- Office hours:
  - On campus: Tuesdays 12:00 noon to 1:30 pm, room 4218 in the STEM center. Entry is from room 4213
  - By appointment on Zoom
- Other resources:
  - Contact Lane via Canvas
  - Online or in-person tutoring via the STEM center (Room 4213)

Check In

- Follow-ups
  - Optical illusion problem
  - Bitwise operators
- Any questions about:
  - Graphics: the canvas, drawing shapes
  - Problems from the homework or extra credit
  - Anything else?
- Please take the Zoom poll!

Flashback to Week 2 ...

Functions

Given a problem to solve:

- Start with the big picture
- Break the problem down into smaller, self-contained building blocks
  - These smaller building blocks are functions
  - The process of breaking down the problem into functions is **decomposition**
  - Any set of steps that will need to be repeated is a good candidate for a function
  - So is any logically self-contained portion of the problem
- Assume the building blocks are done (use `pass` keyword)
- Assemble building in `main` to solve the big problem
- Implement each building block!

How do we get data into or out of a function?

Lecture Review

Functions

- Suppose that in a program, we have to repeatedly check pairs of numbers to see which number is smaller
- Since this action is repeated many times in our program, we want to write a function for this
- Every time **`main()`** has two numbers to compare, it calls **`smaller()`** to do the actual comparison. **`smaller()`** then tells **`main()`** which of the two numbers is smaller
- Problems:
  - How does **`main()`** get the pair of numbers to **`smaller()`**?
  - How does **`smaller()`** get the smaller number back to **`main()`**?

Functions

- To get data in: a function takes in zero or more parameters as input
- To get data out: a function returns zero or one values as output
- E.g., here is a function for returning the smaller of two numbers:

```
def smaller(num1, num2):
    if num1 < num2:
        return num1
    return num2
```

- **num1** and **num2** are parameters or the expected input to the function
- The smaller number of the two is the **return** value or the output from the function

Functions

- The function header has the keyword **`def`**, the function name, and parentheses with the parameter list, followed by a colon :

```
def smaller(num1, num2):
```

- Indented below the header is the function body, which does the required work using the input values:

```
    if num1 < num2:
        return num1
    return num2
```

- The function has zero or more return statements depending on its work. E.g., this one could be designed to just print the smaller number, not return it, in which case we would **`print()`** and not **`return`**.

Functions

- The caller calls the function with the appropriate arguments that match the expected parameters, and does what it needs to do with the return value if there is one:

```
def main():
    first_num = int(input('Enter first number: '))
    second_num = int(input('Enter second number: '))
    smaller_num = smaller(first_num, second_num)
    print(f'The smaller number is {smaller_num}')
```

- Here, the values of **`first_num`** and **`second_num`** are the arguments that are passed in to **`smaller()`**.

Functions

- The function is a black box; the caller does not know how the called function does its work.
- E.g., **`smaller()`** could be defined in these alternative ways:

```
def smaller(num1, num2):
    if num1 > num2:
        return num2
    return num1
```

```
def smaller(num1, num2):
    if num1 <= num2:
        return num1
    return num2
```

- From the standpoint of the calling function, it does not matter what happens inside the called function as long as the result is as expected.

Any Questions?

Functions

- Optional: Compare these two programs that do exactly the same thing
  - `logical_operators.py`
  - `logical_operators_func.py`
- The program that uses a separate function to evaluate and print the passed in arguments is much shorter, clearer, and cleaner than the program that does everything in **`main()`**
- There is also less risk of error in the program that uses a separate function

Section problem: In Range

https://codeinplace.stanford.edu/foothill-cs49/ide/a/inrange
Starter code: https://tinyurl.com/inRangeStarter

In Range

- In **`main()`**, prompt the user for three numbers, **`n`**, **`low`**, and **`high`**
- Call **`in_range()`** to check whether **`n`** is in the range between **`low`** and **`high`**, inclusive
- Back in **`main()`**, print whether **`n`** is in range or not
- User input is in **`blue`**:

```
n: 5
low: 2
high: 8
n is in range!
```

```
n: 5
low: 1
high: 3
n is not in range...
```

In Range

- In **`main()`**, to what type should the user inputs be cast?
- How many parameters should **`in_range()`** expect? Of what type?
- What should **`in_range()`** return? What type is this return value?
- How should **`main()`** use this return value to determine the screen output?

```
n: 5
low: 2
high: 8
n is in range!
```

```
n: 5
low: 1
high: 3
n is not in range...
```

Questions Before We Begin?

Section problem:
Medical Test Simulator

https://codeinplace.stanford.edu/foothill-cs49/ide/a/medicaltestsimulator
Starter code: https://tinyurl.com/medicalTestStarter

Medical Test Simulator

- Given:
  - A population of 10,000 individuals
  - An infection rate of 1% for a disease
  - A diagnostic test with 99% accuracy

... what percentage of positive results from the test will be false positives?

```
Number of people: 10000
Test accuracy: 0.99
Infection rate: 0.01
True positives: 93
False positives: 113
False negatives: 1
True negatives: 9793
54.85436893203883% of positive tests were incorrect
```

User input is in **blue**

Medical Test Simulator

- In **`main()`**, prompt the user for these three numbers:
  - number of people
  - test accuracy
  - infection rate
- In **`simulate_tests()`**:
  - Use the given probabilities and simulate a diagnostic test on each individual
    - Randomly determine whether the individual is infected
    - Randomly determine whether the test is accurate

Medical Test Simulator

- In **`simulate_tests()`**, contd.:
  - Hint: the expression **`random.random() < prob`** evaluates to **`True`** with probability **`prob`**
  - Calculate and print these four values:
    - true positives
    - false positives
    - false negatives
    - true negatives
  - Return the proportion of incorrect positive results
- Back in **`main()`**, print out the percentage of incorrect positive tests

Medical Test Simulator

- In data science, this sort of tally is called a **`confusion matrix`**.

	Diagnosed as infected	Diagnosed as not infected
Actually infected	93 (true positives)	1 (false negative)
Actually not infected	113 (false positives)	9793 (true negatives)

```
Number of people: 10000
Test accuracy: 0.99
Infection rate: 0.01
True positives: 93
False positives: 113
False negatives: 1
True negatives: 9793
54.85436893203883% of positive tests were incorrect
```

This test has good **`recall`**: 93 out of 94 infected individuals were identified (98.94%).

This test has poor **`precision`**: 93 out of 206 individuals identified as infected were infected (45.15%).

Medical Test Simulator

- What type(s) should user input be cast to in **`main()`**?
- How many parameters should **`simulate_tests()`** expect? Of what type(s)?
- How should we calculate the four necessary values of true and false positives and negatives?
- Do we need a loop? If so, for what?
- What is the formula for the **`return`** value, i.e., the proportion of incorrect positive results?
- How can we display this as a percentage in **`main()`**?

Questions Before We Begin?

That's all, folks!

Next up: Animations!