CS 61A Fall 2024

Control, Environment Diagrams

Discussion 1: September 4, 2024

Find your group, and then get started on this worksheet on Berkeley time. It's best if everyone has a way to view this worksheet (on your phone/tablet is fine, but a laptop is better).

If you're in a room with an in-person TA, you can just talk to them. If you're in a room with a remote TA, join this Zoom and talk to the TA there. If you're in the waiting room, that means they are talking to another group, and they'll let you in once that's done.

Now switch to Pensieve, a shared editing app that has this same worksheet in a format that makes it easy to collaborate with your group and ask questions of an AI tutor.

- **Everyone**: Go to pensieve.co, log in with your @berkeley.edu email, and **enter your group number** as the room number (which was in the email that assigned you to this lab).

Once you're on Pensieve, you don't need to return to this page; Pensieve has all the same content (but more features). If for some reason Penseive doesn't work, return to this page and continue with the discussion.

Ice Breaker

Everyone say your name and some activity you enjoy doing outside. For each activity, ask if anyone else in your group likes to do that too. (Optional: after lab you could make plans to go do one of these activities together.)

Then, each person share an expression that people say when they really like something, such as "that's awesome" or "nice one" or "bruh." Each person should try to come up with a different expression. Feel free to ask your group for help if you're stuck. You can even use other languages than English. Then, during the discussion, if someone says or does something well, use your expression!

While and If

Learning to use `if` and `while` is an essential skill. During this discussion, focus on what we've studied in the first three lectures: `if`, `while`, assignment (`=`), comparison (`<`, `>`, `==`, …), and arithmetic. Please don't use features of Python that we haven't discussed in class yet, such as `for`, `range`, and lists. We'll have plenty of time for those later in the course, but now is the time to practice the use of `if` (textbook section 1.5.4) and `while` (textbook section 1.5.5).

Q1: Race

The `race` function below sometimes returns the wrong value and sometimes runs forever.

```
def race(x, y):
    """The tortoise always walks x feet per minute, while the hare repeatedly
    runs y feet per minute for 5 minutes, then rests for 5 minutes. Return how
    many minutes pass until the tortoise first catches up to the hare.

    >>> race(5, 7)  # After 7 minutes, both have gone 35 steps
    7
    >>> race(2, 4) # After 10 minutes, both have gone 20 steps
    10
    """
    assert y > x and y <= 2 * x, 'the hare must be fast but not too fast'
    tortoise, hare, minutes = 0, 0, 0
    while minutes == 0 or tortoise - hare:
        tortoise += x
        if minutes % 10 < 5:
            hare += y
        minutes += 1
    return minutes
```

Find positive integers `x` and `y` (with `y` larger than `x` but not larger than `2 * x`) for which either: - `race(x, y)` returns the wrong value or - `race(x, y)` runs forever

You just need to find one pair of numbers that satisfies either of these conditions to finish the question, but if you want to think of more you can.

Notes: - `x += 1` is the same as `x = x + 1` when `x` is assigned to a number. - 0 is a false value and all other numbers are true values.

The value of `race(x, y)` is incorrect when it is not the **first** time the tortoise passes the hare. Try some small numbers (below 5) to see if you can find a case where `tortoise` has become larger than `hare`, but the expression `tortoise - hare` was not zero when it happened.

If you want to discuss this problem with a TA, just ask (in person or on Zoom).

Q2: Fizzbuzz

Implement the classic *Fizz Buzz* sequence. The `fizzbuzz` function takes a positive integer `n` and prints out a *single line* for each integer from 1 to `n`. For each `i`:

- If `i` is divisible by both 3 and 5, print `fizzbuzz`.
- If `i` is divisible by 3 (but not 5), print `fizz`.
- If `i` is divisible by 5 (but not 3), print `buzz`.
- Otherwise, print the number `i`.

Try to make your implementation of `fizzbuzz` concise.

```
def fizzbuzz(n):
    """
    >>> result = fizzbuzz(16)
    1
    2
    fizz
    4
    buzz
    fizz
    7
    8
    fizz
    buzz
    11
    fizz
    13
    14
    fizzbuzz
    16
    >>> print(result)
    None
    """
    "*** YOUR CODE HERE ***"
```

Be careful about the order of your `if` and `elif` clauses: try first checking if the current number is divisible by both 3 and 5, then check for just divisibility by 3 and just divisibility by 5.

Problem Solving

A useful approach to implementing a function is to: 1. Pick an example input and corresponding output. 2. Describe a process (in English) that computes the output from the input using simple steps. 3. Figure out what additional names you'll need to carry out this process. 4. Implement the process in code using those additional names. 5. Determine whether the implementation really works on your original example. 6. Determine whether the implementation really works on other examples. (If not, you might need to revise step 2.)

Importantly, this approach doesn't go straight from reading a question to writing code.

For example, in the `is_prime` problem below, you could: 1. Pick `n` is 9 as the input and `False` as the output. 2. Here's a process: Check that `9` (`n`) is not a multiple of any integers between 1 and `9` (`n`). 3. Introduce `i` to represent each number between 1 and 9 (`n`). 4. Implement `is_prime` (you get to do this part with your group). 5. Check that `is_prime(9)` will return `False` by thinking through the execution of the code. 6. Check that `is_prime(3)` will return `True` and `is_prime(1)` will return `False`.

Try this approach together on the next two problems.

Important: It's highly recommended that you **don't** check your work using a computer right away. Instead, talk to your group and think to try to figure out if an answer is correct. On exams, you won't be able to guess and check because you won't have a Python interpreter. Now is a great time to practice checking your work by thinking through examples. You could even draw an environment diagram!

If you're not sure about how something works or get stuck, ask for help from the course staff.

Q3: Is Prime?

Write a function that returns `True` if a positive integer `n` is a prime number and `False` otherwise.

A prime number n is a number that is not divisible by any numbers other than 1 and n itself. For example, 13 is prime, since it is only divisible by 1 and 13, but 14 is not, since it is divisible by 1, 2, 7, and 14.

Use the `%` operator: `x % y` returns the remainder of `x` when divided by `y`.

Here's a `while` statement that goes through all numbers above 1 and below `n` :

```
i = 2
while i < n:
    ...
    i = i + 1
```

You can use `n % i == 0` to check whether `i` is a factor of `n`. If it is, `return False`.

```
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    >>> is_prime(1) # one is not a prime number!!
    False
    """
    "*** YOUR CODE HERE ***"
```

Presentation Time: Come up with a **one sentence description** of the process you implemented to solve `is_prime` that you think someone could understand without looking at your code. Share the sentence with your TA for feedback. The purpose of this quick exercise is to practice talking about the behavior of a program using a useful amount of detail.

Q4: Unique Digits

Write a function that returns the number of unique digits in a positive integer.

> **Hints:** You can use `//` and `%` to separate a positive integer into its one's digit and the rest of its digits.
>
> You may find it helpful to first define a function `has_digit(n, k)`, which determines whether a number `n` has digit `k`.

```
def unique_digits(n):
    """Return the number of unique digits in positive integer n.

    >>> unique_digits(8675309) # All are unique
    7
    >>> unique_digits(13173131) # 1, 3, and 7
    3
    >>> unique_digits(101) # 0 and 1
    2
    """
    "*** YOUR CODE HERE ***"









def has_digit(n, k):
    """Returns whether k is a digit in n.

    >>> has_digit(10, 1)
    True
    >>> has_digit(12, 7)
    False
    """
    assert k >= 0 and k < 10
    "*** YOUR CODE HERE ***"
```

One approach is to loop through every digit from 0 to 9 and check whether `n` has the digit. Count up the ones it has.

Environment Diagrams

An **environment diagram** keeps track of names and their values in frames, which are drawn as boxes.

Q5: Bottles

Answer the following questions with your group. Step through the diagram to check your answers.

1) What determines how many different frames appear in an environment diagram?

a) The number of functions defined in the code
b) The number of call expressions in the code
c) The number of return statements in the code
d) The number of times user-defined functions are called when running the code

2) What happens to the return value of `pass_it(bottles)`?

a) It is used as the new value of `remaining` in the global frame
b) It is used as the new value of `bottles` in the global frame
c) It is used as the new value of `pass_it` in the global frame
d) None of the above

3) What effect does the line `bottles = 98` have on the global frame?

a) It temporarily changes the value bound to `bottles` in the global frame.
b) It permanently changes the value bound to `bottles` in the global frame.
c) It has no effect on the global frame.

See the web version of this resource for the environment diagram.

Q6: Double Trouble

Draw the environment diagram on paper or a whiteboard (without having the computer draw it for you)! Then, check your work by stepping through the diagram.

See the web version of this resource for the environment diagram.

Document the occasion

Please all fill out the attendance form (one submission per person per week).