CS 61A Fall 2023

Control, Environment Diagrams

Discussion 1: August 30, 2023

What everyone will need: * A way to view this worksheet (on your phone is fine) * A way to take notes (paper or a device)

What the group will need: * Someone taking notes on a device (perhaps right here on this webpage) who is willing to share their screen of notes with a TA. They should open Discord. (It's fine if multiple people do this, but not everyone has to.)

Please post `Hello, World` in your channel's text chat to let the staff know you made it this far.

NEW: To get help from a TA, we've added a text channel called `discuss-queue`. Any time your group needs help, send a message to `discuss-queue` with the @discuss tag and your discussion group number–a TA will join you shortly.

The purpose of discussion is not just to answer the problems on this worksheet, but to practice so that you can solve these kinds of problems on your own in the future. Use your time together to share what you've learned in the course so far and discuss problem-solving strategies.

Control

Control statements determine which statement or expression is executed next.

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: - `race(x, y)` returns the wrong value - `race(x, y)` runs forever

Challenge: Can you describe in general all of the `x` and `y` that will cause each of these two problems (wrong value and running forever)?

Note: `x += 1` is the same as `x = x + 1` when `x` is assigned to a number.

Put your `(x, y)` examples in the text chat of your group's voice channel. If you get stuck for more than 10 minutes, ask for help from the course staff.

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 ***"
```

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=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.

It's highly recommended that you don't use the Python interpreter to check your work on these questions, at least until after your group is certain that your 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 your solution 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.

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

```
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 ***"
```

Consider talking to the course staff about your solution. They might have tips!

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 ***"
```

Please make sure that everyone in your group understands the logic and implementation before moving on.

Environment Diagrams

When you have questions about how environment diagrams work, ask the course staff!

An **environment diagram** is a model we use to keep track of all the variables that have been defined and the values they are bound to. We will be using this tool throughout the course to understand complex programs involving several different assignments and function calls.

One key idea in environment diagrams is the **frame**. A frame helps us keep track of what variables have been defined in the current execution environment, and what values they hold. The frame we start off with when executing a program from scratch is what we call the **Global frame**. Later, we'll get into how new frames are created and how they may depend on their parent frame.

Here's a short program and its corresponding diagram (only visible on the online version of this worksheet):

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

Assignment Statements

Assignment statements, such as `x = 3`, define variables in programs. To execute one in an environment diagram, record the variable name and the value:

1. Evaluate the expression on the right side of the `=` sign.
2. Write the variable name and the expression's value in the current frame.

Q5: Assignment Diagram

Use these rules to draw an environment diagram for the assignment statements below:

```
x = 11 % 4
y = x
x **= 2
```

def Statements

A `def` statement creates ("defines") a function object and binds it to a name. To diagram `def` statements, record the function name and bind the function object to the name. It's also important to write the **parent frame** of the function, which is where the function is defined.

A very important note: Assignments for `def` statements use pointers to functions, which can have different behavior than primitive assignments (such as variables bound to numbers).

1. Draw the function object to the right-hand-side of the frames, denoting the intrinsic name of the function, its parameters, and the parent frame (e.g. `func square(x) [parent = Global]`.
2. Write the function name in the current frame and draw an arrow from the name to the function object.

Q6: def Diagram

Use these rules for defining functions and the rules for assignment statements to draw a diagram for the code below.

```
def double(x):
    return x * 2

def triple(x):
    return x * 3

hat = double
double = triple
```

Document the occasion

If you'd like, take another group selfie and add it to the text chat of your group's Discord channel. Then, please all fill out the attendance form (one submission per person per week).