DATA C88C
Fall 2025

Control, Python Lists

Discussion 2: September 8, 2025

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`: runs code only when a condition is true - `while`: repeats code as long as a condition is true - assignment (`=`): stores a value in a variable - comparison (`<`, `>`, `==`, …): checks relationships between values - 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: 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
    """
    i = 1
    while i <= n:
        if i % 3 == 0 and i % 5 == 0:
            print('fizzbuzz')
        elif i % 3 == 0:
            print('fizz')
        elif i % 5 == 0:
            print('buzz')
        else:
            print(i)
        i += 1
```

Video walkthrough

Problem Solving

A useful approach to implementing a function is to work step by step— for example, we'll walk through the `is_prime` problem to see how this looks in practice: 1. **Pick an example input and corresponding output.** Pick `n` is 9 as the input and `False` as the output. 2. **Describe a process in English that computes the output from the input.** Here's a process: Check that `9` (`n`) is not a multiple of any integers between 1 and `9` (`n`). 3. **Figure out what additional variables you'll need.** Introduce `i` to represent each number between 1 and 9 (`n`). 4. **Implement the process in code.** Implement `is_prime`. 5. **Test that the implementation works on your original example.** Check that `is_prime(9)` will return `False` by thinking through the execution of the code. 6. **Test that the implementation really works on other examples. (If not, you might need to revise step 2.)** Check that `is_prime(3)` will return `True` and `is_prime(1)` will return `False`.

Important: It's highly recommended that you **don't** check your work using a computer right away. - Instead, talk to people around you and reason it out. - On exams, you won't have access to Python, so practice thinking through examples. - Drawing an environment diagram can help!

This approach doesn't go straight from reading a question to writing code. Try it out on the next two problems. If you're not sure about how something works or get stuck, ask for help from the course staff.

Q2: 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`.

```python
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    >>> is_prime(1) # one is not a prime number!!
    False
    """
    if n == 1:
        return False
    k = 2
    while k < n:
        if n % k == 0:
            return False
        k += 1
    return True
```

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.

Q3: 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
    """
    unique = 0
    while n > 0:
        last = n % 10
        n = n // 10
        if not has_digit(n, last):
            unique += 1
    return unique

# Alternate solution
def unique_digits_alt(n):
    unique = 0
    i = 0
    while i < 10:
        if has_digit(n, i):
            unique += 1
        i += 1
    return unique

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
    while n > 0:
        last = n % 10
        n = n // 10
        if last == k:
            return True
    return False
```

We have provided two solutions: - In one solution, we look at the current digit, and check if the rest of the number contains that digit or not. We only say it's unique if the digit doesn't exist in the rest. We do this for every digit. - In the other, we loop through the numbers 0-9 and just call `has_digit` on each one. If it returns true then we know the entire number contains that digit and we can one to our unique count.