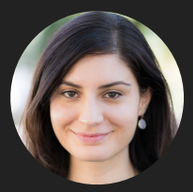
02-01-2023

CS051A

INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

5: while loops



Alexandra Papoutsaki

she/her/hers

Lectures

Zilong Ye

he/him/his

Labs

Welcome to lecture 5, everyone! Are there any questions? How did the second lab go?

Lecture 5: while loops and main function

- prime numbers
- while loops

We will start by looking at prime numbers. What is a prime number?

Prime numbers

- A number that is only divisible by 1 and itself.
- How can we tell if a number is prime?
  - Try and divide it by all of the numbers between 1 and the number.
  - If none of them divide it evenly, then it's prime, otherwise it's not.
    - How can we check to see if a number divides our number evenly?
      - We can use the remainder/modulo operator (%) and see if it equals 0 (i.e. no remainder)
- How can we check all of the numbers?
  - We can use a for loop.

A prime number is a number that is only divisible by 1 and itself. What's an algorithm that we can design to tell if a number is prime? Simple: We can try and divide the **number** by all of the integers between (1,number). If none of them divide it evenly (use the modulo operator and see if the result equals 0), then it's prime. Otherwise it's not. How can we do that? Using a for loop!

isprime_slow function

```
def isprime_slow(num):
    """ Returns True if the input is a prime number, False otherwise """
    for i in range(2, num):
        if num % i == 0:
            return False

    return True
```

```
>>> isprime_slow(5)
True
>>> isprime_slow(6)
False
>>> isprime_slow(100)
False
>>> isprime_slow(101)
True
```

Let’s look at the isprime_slow function in the while.py file. It has a for loop that starts at 2 (instead of 0) and goes up to the number-1. In the past, we saw that the range function takes one parameter and would count from 0 up to but not including the specified number. When we pass 2 parameters, it starts counting at the first number up to, but not including, the second number.
The if statement checks to see if the number is divisible by i. If we find this, we know that the number is not a prime so we don’t have to waste unnecessarily our time. Instead we can return False. Remember, the moment the interpreter sees the return statement it will evaluate the expression that follows and will return it to whomever called the function without executing the rest of its code. What does "return True" do? If we've checked all of the numbers and none of them were divisible (otherwise we would have exited the function with the return False), we know that the number has to be prime and we return True

Prime numbers - a faster version

- Do we need to check all of the numbers up to that number?
  - No, we just need to check up to sqrt(number) (inclusive).

The algorithm we just saw works but there's a faster version. We can check all the numbers between (1, sqrt(number)) instead which saves us steps. Why is that?
If a number n is not a prime, it can be factored into two factors a and b as n = a*b. Now, a and b can't be both greater than sqrt(n), since then the product a*b would be greater than sqrt(n)*sqrt(n) = n. So in any factorization of n, at least one of the factors must be smaller than sqrt(n), and if we can't find any factors less than or equal to sqrt(n), n must be a prime.

isprime function

```python
import math


def isprime(num):
    """ Returns True if the input is a prime number, False otherwise """
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            return False

    return True
```

```
>>> isprime(5)
True
>>> isprime(6)
False
>>> isprime(100)
False
>>> isprime(101)
True
```

Now let’s look at the isprime function in the while.py file. It should feel familiar. It has a for loop that starts at 2 (instead of 0) and goes up to the square root of the number (this is why we need the +1).
The if statement checks to see if the number is divisible by i. If we find this, we know that the number is not a prime so we don’t have to waste unnecessarily our time. Instead we can return False. Remember, the moment the interpreter sees the return statement it will evaluate the expression that follows and will return it to whomever called the function without executing the rest of its code. What does "return True" do? If we've checked all of the numbers and none of them were divisible (otherwise we would have exited the function with the return False), we know that the number has to be prime and we return True

`import math`

- A second way to import a module: `import module_name`
- To reference a function within that module, you then say `module_name.function_name`
- Why might we use this option, i.e. when would we use `from math import *` instead of `import math`?
- Use the first if you're going to be using the functions a lot and it's clear that they come from that module.
- Use the second to be extra clear where the functions are coming from and to avoids naming conflicts.

You might have noticed that the first line in the while.py file was import math. We have seen before that modules allow us to bundle together functions and variables and that we can load them using: from module_name import *. A different way of loading a module is writing import module_name. If we want to use a function now, we would have to write module_name.function_name. This version of loading modules makes extra clear where functions are coming from and avoids naming conflicts.

Print the first x prime numbers

- How could we use `isprime` to print out the first 10 (100, 1000, etc) prime numbers?
  - Some sort of loop? Will a for loop work?
    - We don't know when we're going to stop.
    - We'd like to keep a count of how many we've seen and only stop when we've reached the number we want.

Now let’s say we want to print the first 10 or 100 or 1000 or whatever number prime numbers. How can we use the isprime function to achieve this? We could call it in some sort of loop but would a for loop work? Unfortunately we don’t know when we would have to stop. We would like to keep a count of how many steps we have seen and only stop if we have reached the desired number of prime numbers.

Lecture 5: while loops and main function

- prime numbers
- while loops

How can we do this?

while loops

```
while <bool expression>:
    statement1
    statement2
    ...

statement3
```

- As long as the <bool expression> evaluates to True, it continues to repeat the statements statement1 and statement2. When the expression evaluates to False, it exits the loop and then continues on and executes statement3, etc.

To achieve this, we will use a new type of loop that allows us to do repetition called a for loop. The syntax for it is:

```
while <bool expression>:
    statement1
    statement2
    ...

statement3
```

As long as the <bool expression> evaluates to True, it continues to repeat the statements statement1 and statement2 (and all statements in the while block indicated by indentation). When the expression evaluates to False, it exits the loop and then continues on and executes statement3, etc.

Using a while loop for our prime numbers problem

- Keep a count of how many primes we've found (initially starts at 0).
- Check each number (starting at 2):
  - if it's prime:
    - print it out
    - increment the counter of how many primes we've found
- keep repeating this as long as (while) the number of primes we've printed is less than the number we want.

How can we use a while loop to solve the prime numbers problem we just examined? We can keep a count of how many primes we have found (starting at 0). Starting at 2, we can check one number at a time. If it’s prime (defined by the isprime function), we will print out this number and increment the counter of how many primes we have found. We will repeat this as long as (while!) The number of primes we have printed is less than the number we want.

`firstprimes` function

```
def firstprimes(num):
    """ Prints out the first num primes """
    count = 0  # the number of primes we've printed out
    current = 2  # the current number we're checking

    while count < num:
        if isprime(current):
            count += 1  # same as count = count + 1
            print(current)

        current += 1  # same as current = current + 1
```

That’s exactly what the firstprimes function does in the while.py. current += 1 (means current = current +1) every time through the loop we increment the number we're examining. if that current number happens to be prime, we increment count. the loop continues "while" count < num, that is as long as the number we've found is less than the number we're looking for.

Emulate a while loop as a for loop

- ```
  for i in range(10):
      ...
  ```

  is equivalent to writing:

  ```
  i = 0

  while i < 10:
      ...
      i = i + 1
  ```

Can you emulate a for loop with a while loop? yes!

```
for i in range(10):
    ...
```

is equivalent to writing:

```
i = 0

while i < 10:
    ...
    i = i + 1
```
```

Infinite loops

- ```
  while True:
      print("hello")
  ```
- This code will never stop! It will keep printing hello\nhello\nhello...).
- Sometimes, it will look like the program just froze if you're not actually printing anything out.
- You can stop this by selecting “Stop console” (red square).
- Be careful about these with your program. They're called infinite loops.
- If you think you might have an infinite loop, put in some print statements to debug and think about when the boolean expression will become False and make sure that is going to happen in your loop.

Finally, let’s look at this while loop. It says while True print hello. The code will never stop and it will keep displaying hello messages. Such a loop is called an infinite loop. Sometimes, it will look like the program has frozen and nothing happens. You can stop its execution by selecting Stop console or the red square. If you think your code has an infinite loop, you can debug it using print statements and thinking whether and when your boolean expression will become False so that you can exit the loop.
  ```

Resources

- Textbook: Chapter 8.
- while.py

Practice Problems

- Practice 3 (solution)

Homework

- Assignment 2