COL1000
Introduction to Programming

Priyanka Golia

Most (if not all) of the content is borrowed from Prof. Subodh Kumar's slides

For loops

Given a space-separated list of numbers, find the maximum

```
words = input("enter space sep. numbers")
numbers = words.split()
max = int(numbers[0])
for num in numbers:
    if max < int(num):
        max = int(num)
print("maximum is", max)
```

Initialization

Could have been max = -math.inf
(of course, need to import math first)

Progress/Pattern

For loops

Given a space-separated list of numbers and an integer , check whether integer is present in the list

```
words = input("enter space sep. numbers")
find_num = int(input("enter the number you want to search"))
numbers = words.split()
found = False
for i in range(len(numbers)):
    if find_num == int(numbers[i]):
        print("found at index", i)
        found = True
        break
if not found:
    print(find_num,"is not there in the list")

```

For loops

Given a space-separated list of numbers and an integer , check whether integer is present in the list

```
1  words = input("enter space sep. numbers")
2  find_num = int(input("enter the number you want to search"))
3  numbers = words.split()
4  found = False
5  for num in numbers:
6      if find_num == int(num):
7          print("found")
8          found = True
9          break
10 if not found:
11     print(find_num,"is not there in the list")
```

Break is to break out of loop and jump to after the loop statements

For loops

Test if a given number n is a prime number, n > 1

```
1  num = int(input("enter a number > 1"))
2  if num == 2:
3      print("its a prime number")
4  else:
5      found = 1
6      for i in range(2,num):
7          if not num % i:
8              print("it is not a prime number")
9              found = 0
10             break
11     if found:
12         print("its a prime number")
```

Python will keep executing the loop until the condition becomes false. Sometimes, however, you want to stop the loop early, even if the condition is still true.
That's exactly what `break` does — It immediately stops the loop and jumps to the first statement after the loop

Break is to break out of loop and jump to after the loop statements

Loops and Branches can be nested to any degree

For loops

Test if a given number n is a prime number, n > 1

```
1  num = int(input("enter a number > 1"))
2  if num == 2:
3      print("its a prime number")
4  else:
5      found = 1
6      for i in range(2,num):
7          if not num % i:
8              print("it is not a prime number")
9              found = 0
10             break
11     if found:
12         print("its a prime number")
```

Does this loop have unnecessary iterations?

A simple improvement is to check only up to $\sqrt{n}$. If a number has a factor greater than $\sqrt{n}$, the corresponding smaller factor would already have been found earlier. So we can stop the loop much earlier.

Important question for measuring efficiency.

For loops

Given an integer num, print all the prime numbers between 1 and n (inclusive).

```
1  num = int(input("enter a number"))
2  for n in range(1,num+1):
3      if n <= 1:
4          print(n, "is not prime")
5      elif n == 2:
6          print("2 is prime")
7      else:
8          found = True
9          for i in range(2,n):
10             if not n%i:
11                 print(n, "is not prime")
12                 found = False
13                 break
14         if found:
15             print(n, "is prime")
```

Outer for loop — updated the value of n

Inner for loop — checks if n is prime or not!

Loops and Branches can be nested to any degree

For loops

Given an integer num, print all the prime numbers between 1 and n (inclusive).

```python
num = int(input("enter a number"))
for n in range(1,num+1):
    if n <= 1:
        print(n, "is not prime")
    elif n == 2:
        print("2 is prime")
    else:
        found = True
        for i in range(2,n):
            if not n%i:
                print(n, "is not prime")
                found = False
                break
        if found:
            print(n, "is prime")
```

```
enter a number20
1 is not prime
2 is prime
3 is prime
4 is not prime
5 is prime
6 is not prime
7 is prime
8 is not prime
9 is not prime
10 is not prime
11 is prime
12 is not prime
13 is prime
14 is not prime
15 is not prime
16 is not prime
17 is prime
18 is not prime
19 is prime
20 is not prime
```

For loops

Given an integer num, print all the prime numbers between 1 and n (inclusive).

```
1  num = int(input("enter a number"))
2  prime_numbers = []
3  for n in range(1,num+1):
4      if n <= 1:
5          continue
6      elif n == 2:
7          prime_numbers.append(n)
8      else:
9          found = True
10         for i in range(2,n):
11             if not n%i:
12                 found = False
13                 break
14         if found:
15             prime_numbers.append(n)
16 print("prime numbers between 1 and ", n, "are", prime_numbers)
```

Whenever Python sees a continue, it jumps back to the top of the loop and checks the next value — without executing any of the remaining statements in that iteration.

Break stops the loop completely, Continue just skips one iteration.

Adding n to the list using “append”.

```
enter a number20
prime numbers between 1 and  20 are [2, 3, 5, 7, 11, 13, 17, 19]
```

Recognize Common Errors

- Syntax Error

```
'hello' = False
```

```
four = twice two
```

- NameError

```
five = two + three
```

without earlier giving names two or three to any object

- ValueError

```
int('hello')
```

- TypeError

```
round('a')
```

```
'hello' - 'priyanka'
```

- IndexError

```
words = []
print(words[0])
```

Empty list!

```
words = []
words[0] = 3
```

That is NOT how you add objects to a list.
Lists are "mutable," i.e., an object itself may be changed (unlike strings). But not in this way — To modify i^{th} member, there must be one. (X.append(m) will add m to list named X)

```
words.append(10)
words[0] = 3
```

Effectively, word[0] becomes a way to refer to a new value:

- ZeroDivisionError

```
2.0/0
```