COL1000
Introduction to Programming

Priyanka Golia

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

Nested For loop

```
1 for i in range(1,3):
2     for j in range(1,3):
3         print("i:",i,"j:",j)
```

Outer for loop

Inner for loop

All loops must terminate

```
i: 1 j: 1
i: 1 j: 2
i: 2 j: 1
i: 2 j: 2
```

For the same value of i, values of j is updating

Outer loop: $i = 1$

Inner loop: $j = 1$

$..= 2$

Outer loop: $i = 2$

Inner loop: $j = 1$

$..= 2$

Nested For loop

```
for i in range(0,3):
    for j in range(0,3):
        print(i*j)
```

```
0
0
0
0
1
2
0
2
4
```

By default, print adds a new line after each output. That's why, inside a loop, every value is printed on a new line. If you want to change this, you can use the end option in print to decide what should come after each value.

```
for i in range(0,3):
    for j in range(0,3):
        print(i*j, end = "@")
```

```
0@0@0@0@1@2@0@2@4@
```

```
for i in range(0,3):
    for j in range(0,3):
        print(i*j, end="@")
    print("")
```

```
0@0@0@
0@1@2@
0@2@4@
```

To add newline at the end of inner loop.

While loops

What if we can't know in advance, how many iterations to execute?

Compute the sum of numbers until the user enters o or a negative number:

```
1 n = int(input("enter a number"))
2 sum = 0
3 while n > 0:
4     sum += n
5     print("sum of numbers entered so far:", sum)
6     n = int(input("enter another number"))
7 print("you have now entered zero or a negative number")
8 print("sum of numbers ", sum)
```

Explicitly assigning loop control variable.

Termination condition (loop test) on loop control variable. Could be complex.

Explicitly updating loop control variable.

While loops

```
1 n = int(input("enter a number"))
2 sum = 0
3 while n > 0:
4     sum += n
5     print("sum of numbers entered so far:", sum)
6     n = int(input("enter another number"))
7 print("you have now entered zero or a negative number")
8 print("sum of numbers ", sum)
```

```
enter a number8
sum of numbers entered so far: 8
enter another number7
sum of numbers entered so far: 15
enter another number12
sum of numbers entered so far: 27
enter another number03
sum of numbers entered so far: 30
enter another number2
sum of numbers entered so far: 32
enter another number-2
you have now entered zero or a negative number
sum of numbers  32
```

While loops

Test if a given number n is a prime number

```
1 n = int(input("enter a number"))
2 div = 2
3 while n % div and div < n:
4     div += 1
5 if div == n:
6     print(n, "is a prime number")
7 else:
8     print(n, "is not a prime number")
```

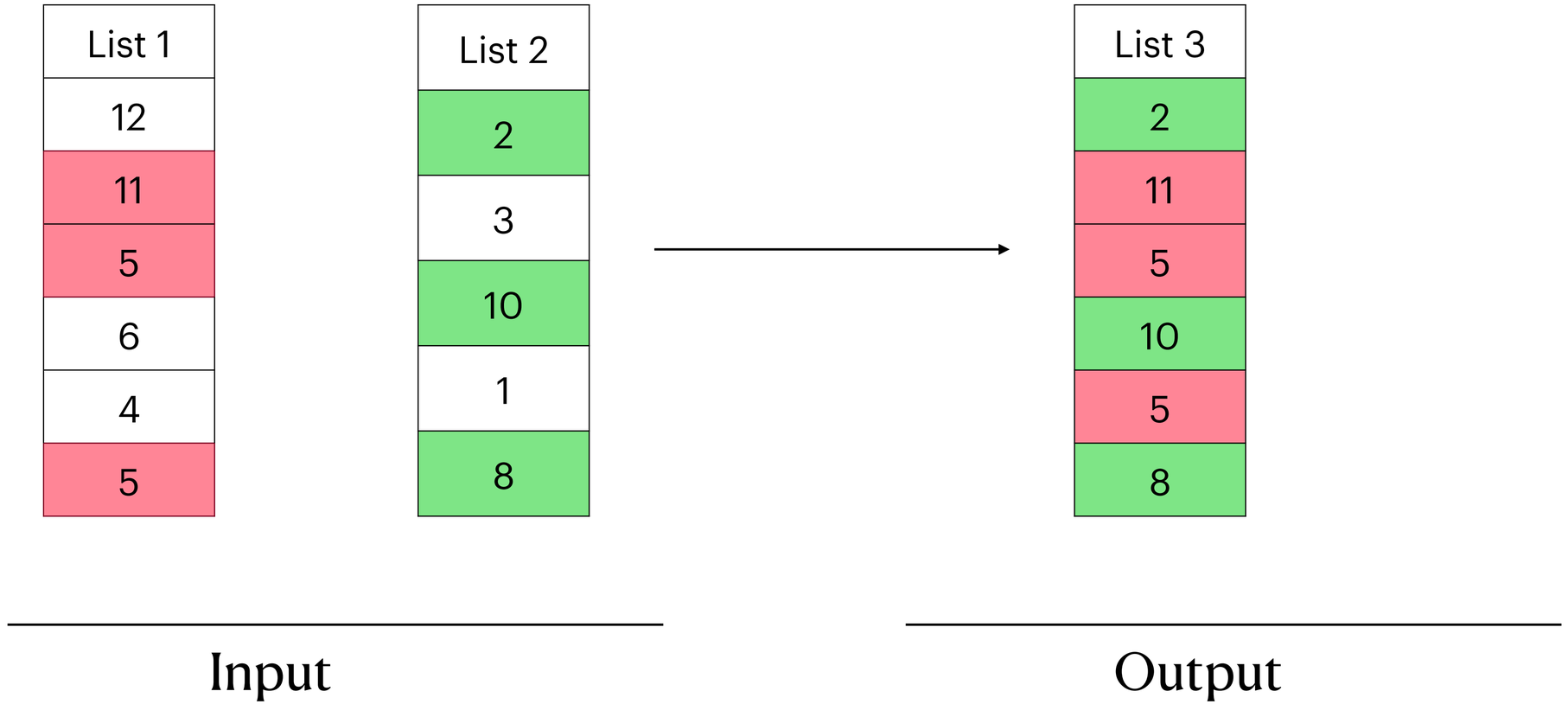
Explicitly assigning loop control variable.

The loop test incorporates the "break" test as well. We didn't need to use "found" like we had in for-loop

Explicitly updating loop control variable.

While loops

Process two list and create a new list. New list should have odd elements of list 1 and even elements of list 2



While loops

Process two list and create a new list. New list should have odd elements of list 1 and even elements of list 2

```
1  list1 = [12,11,5,6,4,5]
2  list2 = [2,3,10,1,8]
3  print(list1)
4  print(list2)
5  l1 = 0
6  l2 = 0
7  list3 = []
8  while l1 < len(list1) or l2 < len(list2):
9      if list1[l1]%2:
10         list3.append(list1[l1])
11     elif not list2[l2] % 2:
12         list3.append(list2[l2])
13     l1 += 1
14     l2 += 1
15 print("list 3", list3)
```

Ln 16, Col 1 Python

```
[12, 11, 5, 6, 4, 5]
[2, 3, 10, 1, 8]
list 3 [2, 11, 5, 8, 5]
```

Notice index 2, that is, $list1[2] = 5$, and $list2[2] = 10$. In $list3$, the object 10 didn't appear.

It is because, we have the condition to check if $list2'$ element is even in the "elif". If the ``if" condition evaluates to True, the "elif" block is skipped and does not get executed.

A program being "correct" does not just mean it runs without syntax errors. That is only the first step. A program is correct when it actually does what we want it to do — meaning, for every possible input, it should give the right output.

While loops

Process two list and create a new list. New list should have odd elements of list 1 and even elements of list 2

```
1  list1 = [12,11,5,6,4,5]
2  list2 = [2,3,10,1,8]
3  print(list1)
4  print(list2)
5  l1 = 0
6  l2 = 0
7  list3 = []
8  while l1 < len(list1) or l2 < len(list2):
9      if list1[l1]%2:
10         list3.append(list1[l1])
11     if list2[l2] % 2:
12         list3.append(list2[l2])
13     l1 += 1
14     l2 += 1
15 print("list 3", list3)
```

```
[12, 11, 5, 6, 4, 5]
[2, 3, 10, 1, 8]
Traceback (most recent call last):
  File "run.py", line 1, in <module>
    import lec_main
  File "/home/p11376/lec_main.py", line 3, in <module>
    import lec11
  File "/home/p11376/lec11.py", line 11, in <module>
    if list2[l2] % 2:
IndexError: list index out of range
```

An **IndexError** occurs when you try to access a position in a list, string, or any other sequence type that does not exist.

Example: numbers = [1, 2, 3]

print(numbers[3]) # ❌ IndexError

Recall that indexing in Python starts at 0. So the valid indexes here are 0, 1 and 2. There is no element at index 3, which is why this statement gives an **IndexError**.

While loops

Process two list and create a new list. New list should have odd elements of list 1 and even elements of list 2

```
1  list1 = [12,11,5,6,4,5]
2  list2 = [2,3,10,1,8]
3  print(list1)
4  print(list2)
5  l1 = 0
6  l2 = 0
7  list3 = []
8  while l1 < len(list1) or l2 < len(list2):
9      if list1[l1]%2:
10         list3.append(list1[l1])
11     if list2[l2] % 2:
12         list3.append(list2[l2])
13     l1 += 1
14     l2 += 1
15 print("list 3", list3)
```

```
[12, 11, 5, 6, 4, 5]
[2, 3, 10, 1, 8]
Traceback (most recent call last):
  File "run.py", line 1, in <module>
    import lec_main
  File "/home/p11376/lec_main.py", line 3, in <module>
    import lec11
  File "/home/p11376/lec11.py", line 11, in <module>
    if list2[l2] % 2:
IndexError: list index out of range
```

Here, the value of the loop variable $l2$ increases with each iteration. However, since $len(list1) > len(list2)$, the program eventually tries to access an element outside the valid index range of $list2$—specifically $list2[5]$.

While loops

Process two list and create a new list. New list should have odd elements of list 1 and even elements of list 2

```
1  list1 = [12,11,5,6,4,5]
2  list2 = [2,3,10,1,8]
3  print(list1)
4  print(list2)
5  l1 = 0
6  l2 = 0
7  list3 = []
8  while l1 < len(list1) or l2 < len(list2):
9      if l1 < len(list1) and list1[l1]%2:
10         list3.append(list1[l1])
11     if l2 < len(list2) and not list2[l2] % 2:
12         list3.append(list2[l2])
13     l1 += 1
14     l2 += 1
15 print("list 3", list3)
```

```
[12, 11, 5, 6, 4, 5]
[2, 3, 10, 1, 8]
list 3 [2, 11, 5, 10, 8, 5]
```

Checking if index is less than the length of list — to fix the index error.

Infinite Loop

An **infinite loop** is a loop that **never ends**.
It keeps repeating forever because the **stopping condition is never met** (or sometimes missing).

```
1 n = 5
2 while n >= 5:
3     print("hello")
4
```

We are not updating the value of n. It is always 5. While condition is always True.

```
1 n = 5
2 while True:
3     print("hello")
4     if n > 5:
5         break
6
```

We can have conditions like this — no syntax error.

But, this will cause infinite loop, as again, we are not updating the value of n. So, n will never be greater than 5, and “break” will not get executed.

Nested while loop

Given an integer n and print the multiplication tables for all numbers from 2 to n.

```
1 n = int(input("enter a number"))
2 while n > 1:
3     i = 1
4     print(f"{n}'s table")
5     while i <= 5:
6         print(f"{n} x {i} = {n*i}")
7         i += 1
8     n -= 1
```

Outer loop

Inner loop

Inner loop controlling variable

Outer loop controlling variable

All loops must terminate

```
enter a number3
3's table
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
2's table
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
```

Recall — f-string (f"...") allows you to directly embed variables inside curly braces {}.

For vs While loops

Aspect	`for` loop	while loop
When to use	When the number of iterations is known in advance. Also, called us "**Definite**" loops	When repeating until a condition becomes false. Also, called us "**Indefinite**" loops
Control variable	Automatically handled by Python (ex. `range`)	Must be defined and updated manually
Condition check	Implicitly checked every time through (ex. `range`)	Explicitly written in the `while` condition
Risk of infinite loop	low	High if the condition never becomes false or the variable is not updated
Example use case	Printing numbers 1 to 10, iterating over a list	Keep asking user for input until valid, sum until zero is entered
Syntax emphasis	for i in range(...):	while <condition>:

More Examples!

Check Prof. Sayan's slides

1. Write a program that reads an integer from the user and prints the reverse of the number. For example:

If the input is 5629 the output should be 9256.

If the input is 120, the output should be 021.

Note: Do not convert the number to a string. Use integer arithmetic and a loop.

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

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`