



# CS 1112: Introduction To Programming

Loops: While-loops and For-loops

Dr. Nada Basit // `basit[at]Virginia[dot]edu`

# Friendly Reminders

---

- Your **safety** and **comfort** is important!
  - If you choose to wear a mask you are welcome to do so
  - *We will interpret wearing a mask as being considerate and caring of others in the classroom (not that you are sick), and realize that some may choose to mask to remain distanced*
- Remember to always be **kind, respectful, supportive, compassionate** and **mindful of others!** 😊
- Be an **active** participant in your learning!  
You're welcome and **encouraged** to ask questions during class!
- If you feel **unwell**, or think you are, **please stay home**
  - *Contact us! We will work with you!*
  - Get some rest 😊
  - View the recorded lectures – *please allow 24-48 hours to post*



# Announcements

---

- **Quiz 4** is **due by 11:00pm on Monday (tonight)**!
  - No late quizzes accepted
  - No make-up quizzes allowed
  - If you believe your computer is glitching, it's a good idea to copy down your answers to each of the questions in a word document. In the event something happens, you can send me your solutions.
  - ***Note:** in general, will **cannot and will not** accept quiz solutions via **email**. We will only accept them in the case where your quiz may have glitched and we **no longer have your submitted answers**.*
  - **Take quiz on:** [Sherlock.cs.virginia.edu](https://sherlock.cs.virginia.edu)
- **PA03** is **due by 11:00pm on Wednesday (Feb. 19)**!
  - Submit on Gradescope: your .py file
- **Exam 1** is coming up... on **February 26, 2025**!
  - If you have **SDAC** time and/or distraction-free accommodations, please **book** a time slot with SDAC to take the exam **at their facility**. Book **any time** on Feb. 26<sup>th</sup>!

# Earlier in the Semester We Mentioned the Building Blocks of Programs

---

- **Sequence**

- We start with the instruction written at the top
- We go in order, one instruction at a time
- Each line is “one” thing to do

- **Repetition** – repeat something

- Repeat a fixed number of times (e.g., repeat 5 times)
- Repeat until something happens (e.g., repeat until input is valid)

- **Conditions/Decisions** – maybe do something

- Check something first, i.e., if there is a file present, read it

- **Named actions**

- Grouping many lower-level actions in to one higher level name
  - Definition of the name action
  - Use of the named action

← We have been introduced to for-loops earlier, so we will start with **while-loops** and then review for-loops!

# Conditional Decision Statement

- **Recall:** To define code that sometimes runs:

```
if boolean expression:  
    statements  
elif boolean expression:  
    statements
```

Remember  
Boolean  
expressions!

1 of these

Boolean expressions are  
also used in "while" loops!!

# Agenda

---

- While-loops
- For-loops (*we've seen before, but we will formalize*)
- Contrast these two kinds of loops

---

# While-loops

# While loops

---

- Define code that runs **until** a **condition** is **False**

```
while boolean_expression:  
    statement(s)
```

Guard Condition



Keeps doing the action (execute statements in body) over and over as long as the boolean expression is **True**.

- Example:

```
x = 147  
while x >= 100:  
    sub = int(input('Enter a number to subtract from x: '))  
    x = x - sub  
    . . .
```



# ★ While Loops

The **condition** of the while loop is checked **BEFORE** the subsequent iteration.

If the condition is **TRUE**, then the code inside the loop is executed.

If the condition is **FALSE**, the loop stops.


```
i = 0
while i < 5:
    print("Hello World (Example 1)")
    i+=1
```

1. A while loop evaluates the **Boolean condition** (“*Guard condition*”)
2. If the condition evaluates to **True**, **all** the code inside the while loop is *executed* [once]
3. The condition is *evaluated* again
4. This process continues **until** the condition is **False**
5. When the condition evaluates to **False**, the loop **stops**

# ★ Rules for While Loops

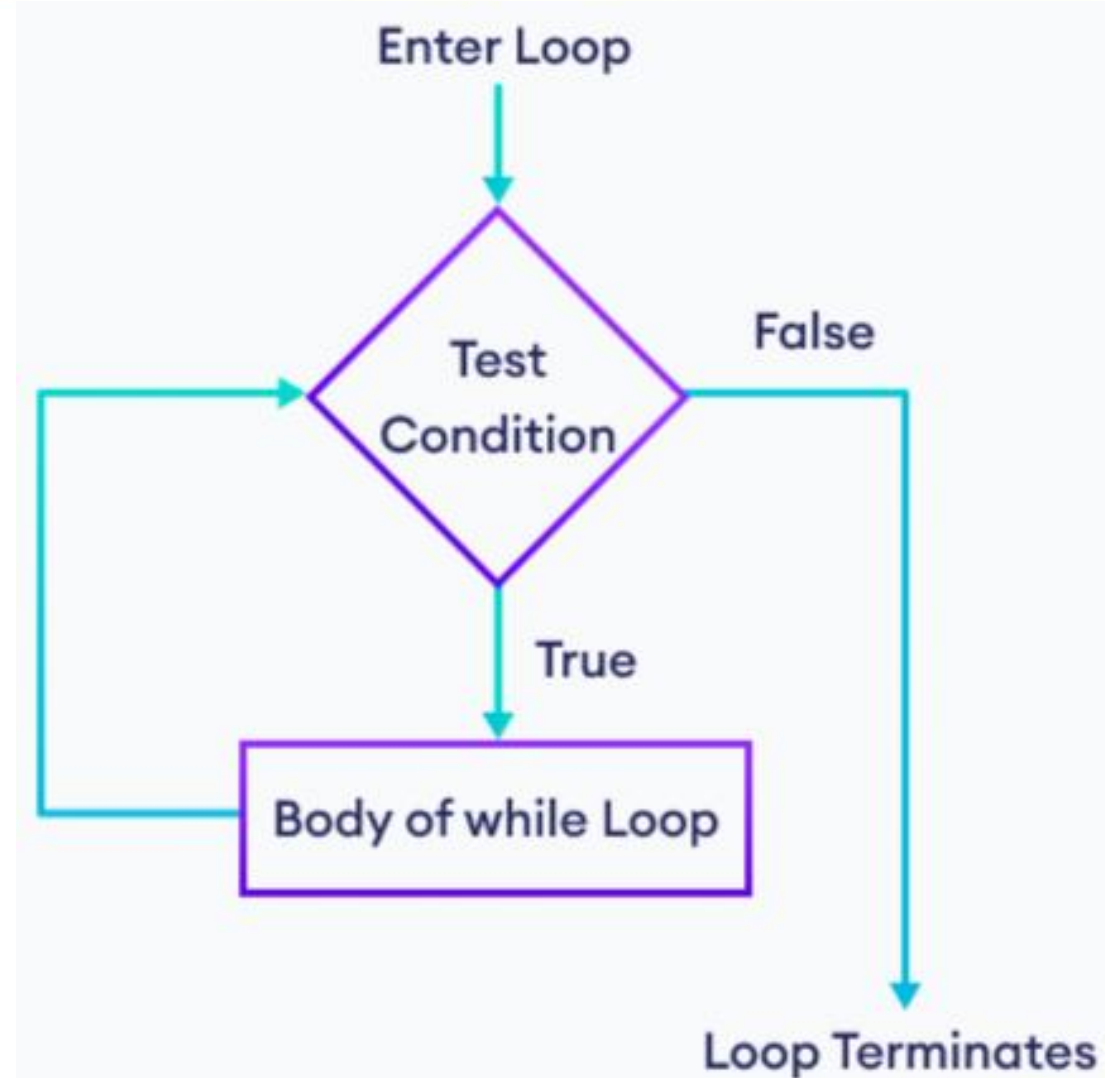
---

```
i = 0
while i < 5:
    print("Hello World (Example 1)")
    i+=1
```



- Some aspect of your *guard condition* must *change* in the *body*
- The *condition* must *change* in such a way that the Boolean expression will eventually become **False**
- Every statement in the body of the while loop is finished *before checking the guard again*
- **Note:** The guard will only be checked after we get to the **end** of the body of the while loop, *not after each statement*

# ★ Flowchart of Python While-loop



```
# initialize the variable
i = 1
n = 5

# while loop from i = 1 to 5
while i <= n:
    print(i)
    i = i + 1
```

## Worked Example: While-loop

Variable	Condition: <code>i &lt;= n</code>	Action
<code>i = 1</code>  <code>n = 5</code>	True	1 is printed. <code>i</code> is increased to 2.
<code>i = 2</code>  <code>n = 5</code>	True	2 is printed. <code>i</code> is increased to 3.
<code>i = 3</code>  <code>n = 5</code>	True	3 is printed. <code>i</code> is increased to 4.
<code>i = 4</code>  <code>n = 5</code>	True	4 is printed. <code>i</code> is increased to 5.
<code>i = 5</code>  <code>n = 5</code>	True	5 is printed. <code>i</code> is increased to 6.
<code>i = 6</code>  <code>n = 5</code>	False	The loop is terminated.

# Q1: How many times will this run?

---

```
x = 0
```

```
while x > 5:
```

```
    print(x)
```

Q1: How many times will this run? **Zero (0) times**

---



```
x = 0
```

```
while x > 5: # x starts out by being < 5, so the condition is False  
    print(x)
```

## Q2: How many times will this run?

---

```
x = 0
```

```
while x < 5:
```

```
    print(x)
```

## Q2: How many times will this run? Infinite



```
x = 0
```

```
while x < 5:    # x is less than 5, but it NEVER CHANGES... so infinite!  
    print(x)
```



### Q3: How many times will this run?


---

```
x = 0
```

```
while x < 5:
```

```
    print(x)
```

```
    x += 1
```



# Q3: How many times will this run? 5 times



```
x = 0
```

```
while x < 5:
```

```
    print(x)
```

```
    x += 1 # the guard condition IS changed in the body of the loop  
           # (eventually the guard condition will become FALSE)
```

# Repetition with incrementing

---

```
x = 0
while x < 5:
    print(x)
    x += 1
```

This repeats the code inside the **while** loop body **five** times:

- The first time through the loop, **x = 0**
- The second time through the loop, **x = 1**
- The third time through the loop, **x = 2**
- The fourth time through the loop, **x = 3**
- The fifth time through the loop, **x = 4**
- We do **\*not\*** repeat the loop when **x becomes 5**

# Q4: What is the last line in the body of this while loop?

---

```
2     target = 77
3     count = 10
4     while count > 0:
5         z = int(input("Enter a number: "))
6         if z == target:
7             print("You win a prize!")
8         else:
9             print(str(count - 1), "left")
10        count -= 1
11    print("Program finished")
```

## Q4: What is the last line in the body of this while loop?

---

```
2     target = 77
3     count = 10
4     while count > 0:
5         z = int(input("Enter a number: "))
6         if z == target:
7             print("You win a prize!")
8         else:
9             print(str(count - 1), "left")
10        count -= 1
11    print("Program finished")
```

**Line 10**

Body of While Loop

# Other Kinds of While Loops

---

- There is also a
  - **While-Else loop**
  - A version of a **Do-While loop**
- We'll see examples of these in the Python file

---

# PYTHON DEMONSTRATION

Let's jump on PyCharm!

`while_loops.py`

```

# while loops

# Example - Are we there yet?
# Keep checking until we are there....
#
text = input("Are we there yet? ") # what happens if we do not give
answer

                                # a value before the loop?
while text == 'no': # the condition is - <text == 'no'>
    print("Whatever...")
    text = input("Are we there yet? ") # but if we changed text to text1?

print("Program Finished")


# use a while loop to count up
#
# How many dozen eggs do I need to buy?
#
eggs_bought = 0
target = int(input("How many eggs do you need? "))
while (eggs_bought * 12) < target:
    eggs_bought += 1

print("You should buy", eggs_bought, "dozen")

```

```

# This is called an *Input Validation* Loop

# *force* the user to enter a number from 1 to 100

number = int(input("Enter a number from 1 and 100: "))
# Inclusive: we want numbers >= 1 but <= 100
while number < 0 or number > 100:
    number = int(input("That number won't work. Try again: "))

print("Okay, your number was", number)

```

Review this code on your own.  
Don't hesitate to ask the TAs or the  
professor questions if you have any!



```

# Example - Countdown timer
# 10-1 Blastoff
print("COUNTDOWN STARTING")
current_num = 10
while current_num > 0:
    print(current_num, '.....')
    current_num-=1
print('BLASTOFF!!!')

# Now create a function that counts down from a given number to 0.
# Show each number as the countdown happens, at 0 print "Blastoff!"
#
def countdown(seconds):
    # What condition should we use?
    while seconds > 0: # This is better than using 'seconds != 0' Why?
        print(seconds)
        seconds-=1 # what if we used seconds+=1
    print("Blastoff!!!")

countdown(10)
print("Program Finished")

```

```

# Example - Fizzbuzz
# Look at every number in a given range to see which are fizzbuzz numbers.
# if the number is divisible by 3, print 'fizz'
# if the number is divisible by 5, print 'buzz'
# if the number is divisible by both, print 'fizzbuzz'
#
def fizzbuzz(x):
    while x >= 0: #
        print(x, " ",end="") # print out each number as we count down
        if x % 3 == 0: # if the number is divisible by 3
            print("fizz",end="") # don't go to a new line yet
        if x % 5 == 0:
            print("buzz",end="")
        print() # move the cursor to the next line
        x-=1 # What if we move this back to the left one indentation level?

fizzbuzz(30)
print("Program Finished")


```

Review this code on your own.  
Don't hesitate to ask the TAs or the professor questions if you have any!

---

# For-loops





Remember  
me?

## We've Seen For-loops Before!

- We saw **for-loops** in the context of **Turtles**!
- We didn't formally cover for-loops, but we showed them in use, and all of you wrote code using them!
- Today we will cover for-loops more formally, **so most of this section should be a review!**

# ★ For-loops

---

- Define code that runs once for each thing in a collection

```
for <variable> in <collection>:  
    statements
```

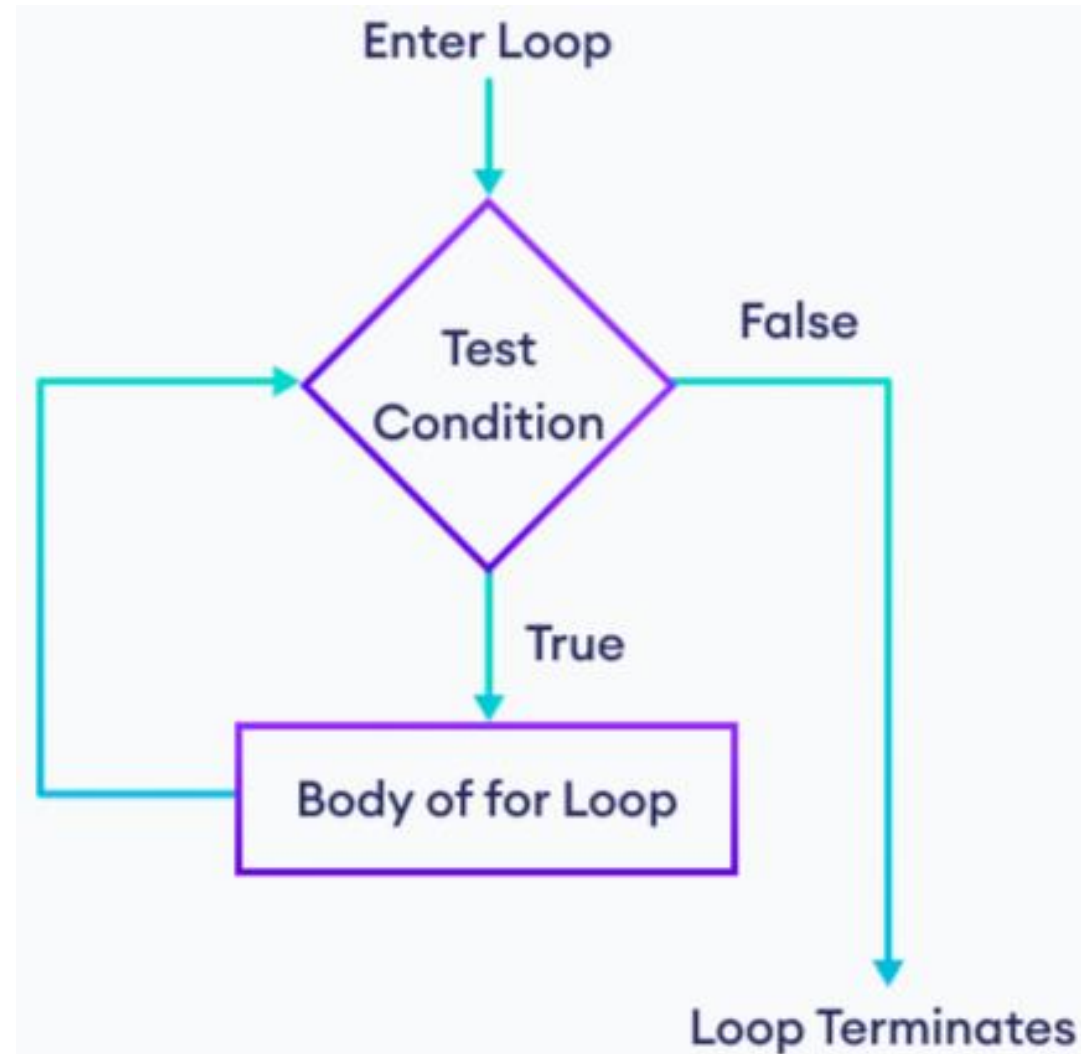
**strings** -       for each in "desk":  
                  print(each)

**lists** -           for item in ['apple', 'banana', 'orange']:  
                  print(item)

**integers** -       for i in range(5):  
                  print(i, 'hi')



# ★ Flowchart of Python For-loop



# Worked Example: For-loop

```
# use of range() to define a range of values
values = range(4)

# iterate from i = 0 to i = 3
for i in values:
    print(i)
```

The value of `i` is set to **0** and it is updated to the next number of the range on each iteration. This process continues until **3** is reached.

Iteration	Condition	Action
1st	True	0 is printed. <code>i</code> is increased to 1.
2nd	True	1 is printed. <code>i</code> is increased to 2.
3rd	True	2 is printed. <code>i</code> is increased to 3.
4th	True	3 is printed. <code>i</code> is increased to 4.
5th	False	The loop is terminated

# Looping through integers - using the range() function

---

- **range(stop):**
  - Gives all integers from 0 (inclusive) to stop (exclusive)
  - range(5) -> 0, 1, 2, 3, 4
- **range(start, stop):**
  - Gives all integers from start (inclusive) to stop (exclusive)
  - range(2,6) -> 2, 3, 4, 5
  - start defaults to 0
- **range(start, stop, step):**
  - Gives all integers from start (inclusive) to stop (exclusive), but it takes only every step item
  - The 3rd argument is the step-size, or increment size. It defaults to 1 (increase by 1).

See Supplemental slides at the end of this presentation for additional information about for-loops and range()!

**Note** that range(start, stop) and range(start, stop, step) behave similarly to string slicing

# Repetition with incrementing

---

```
total = 0
for count in range(5):
    total = total + count
print(total)
```

- This repeats the code inside the **for** loop body **five times**:
  - The first time through the loop, `count = 0`, `total = 0`
  - The second time through the loop, `count = 1`, `total = 1`
  - The third time through the loop, `count = 2`, `total = 3`
  - The fourth time through the loop, `count = 3`, `total = 6`
  - The fifth time through the loop, `count = 4`, `total = 10`
- We do **\*not\*** repeat the loop when `count = 5`



# Let's See What We Can Remember:

## What is printed?

---

```
x = "123"
```

```
for i in x:
```

```
    print("a")
```

# Let's See What We Can Remember:

## --Answer:

---

```
x = "123"
```

```
for i in x:
```

```
    print("a")
```



a

a

a

# Let's See What We Can Remember:

## What is printed?

---

```
x = "123"  
  
for i in x:  
    print()  
  
print("a")
```

# Let's See What We Can Remember:

--Answer:

---

```
x = "123"
```

```
for i in x:
```

```
    print()
```

```
print("a")
```

```
<blank line>
```

```
<blank line>
```

```
<blank line>
```

```
a
```

We will actually see nothing, but I'm using **<blank line>** to indicate that a blank line is created here!

Output:

Three blank lines followed by the letter a

# Let's See What We Can Remember:

## What is printed?

---

```
total = 0
for count in range(1, 5):
    total += count
print(total)
```

# Let's See What We Can Remember:

## --Answer:

---

```
total = 0
```

```
for count in range(1,5): # 1, 2, 3, 4
```

```
    total += count # 1 + 2 + 3 + 4 = 10
```

```
print(total)
```

10

# Let's See What We Can Remember:

## What is printed?

---

```
total = 0
for count in range(1,10):
    double = count * 2
    total = total + double
print(total)
```

# Let's See What We Can Remember:

--Answer:

---

```
total = 0
for count in range(1,10): # 1, 2, 3, 4, 5, 6, 7, 8, 9
    double = count * 2
    total = total + double # 2 + 4 + 6 + 8 + 10 + 12 + 14 + 16 + 18
print(total)
```

90



## ★ For-loop with Lists

---

```
dogs = ["stewart", "apollo", "bolt", "mitsy", "maggie"]
for name in dogs:
    print(name, type(name))
    go_to_vet(name) # Call the go_to_vet() function
print("Program finished")
```

---

```
temperature = [83.4, 78.3, 87.2]
for current_temp in temperature:
    if current_temp > 80:
        print("It's hot", current_temp)
    else:
        print("It's only warm", current_temp)

print(temperature)
```

# ★ Accumulator pattern

---

- Idea: loop through some **collection** and “**accumulate**” some stuff
- Start with **nothing** (*initialize the accumulator variable*)
- **Repeat:**
  - Add to it (*modify the accumulator variable*)
- **Done:** the accumulator has all of your stuff

```
total = 0
for count in range(1,101):
    total = total + count
print(total)
```

```
vowels = ''
for letter in 'Guido is my hero':
    if letter in 'aeiou':
        vowels = vowels + letter
print(vowels)
```

# Please Review Supplemental Material

---



- For additional explanation and examples please review the **supplemental material** at the end of these slides.
- **Topics covered include:**
  - Range() function (*including an online resource*)
  - For loop and range()
  - Augmented Assignment Operators (*\*please take note of these!\**)
  - Practice questions (*and answers*)

# Practice Problem: Average Rainfall

---

- You have several numbers representing daily rainfall over the course of several days
- Calculate the average daily rainfall
- A value of 0 means that no rain was recorded that day
- Note: On some days the sensors *failed* and recorded a **negative number**. **Don't** use those days as part of the average.



**data** = [0, 1.3, 2.2, -565, 0, 16, -2.1, 0, 2.1]



**# *Rainfall problem:***

- # - *Given a list of values representing daily rainfall*
- # - *Calculate average daily rainfall*
- # - *Disregard negative values (faulty/incorrect data)*

```
data = [0, 1.3, 2.2, -565, 0, 16, -2.1, 0, 2.1]
```

```
total_rainfall = 0
```

```
total_items = 0
```

```
for item in data:
```

```
    if item >= 0: # disregard negative values
```

```
        total_rainfall = total_rainfall + item
```

```
        total_items = total_items + 1
```

```
print(total_rainfall/total_items)
```

---

# Comparison

# ★ Recap on Loops ★

**for** loops behave like blocks of statements that have been copied and pasted a certain number of times

```
for thing in collection:  
    Do stuff (probably use thing)
```

```
thing = collection[0]  
Do stuff (probably use thing)  
thing = collection[1]  
Do stuff (probably use thing)  
thing = collection[2]  
Do stuff (probably use thing)
```

**while** loops behave like lots of if statements

```
while boolean_expression:  
    Do stuff (expression values should change)
```

```
if boolean_expression:  
    Do stuff  
if boolean_expression:  
    Do stuff  
if boolean_expression:  
    Do stuff
```



# When to use each kind of loop

---

## For

- We know **how many times** we **should repeat something**
- We want to do something **for each item in a collection**

## While

- We only know **what** should make **us stop**
- We want to continue doing something **under certain circumstances**



---

# PYTHON DEMONSTRATION

Let's jump on PyCharm!

`for_loops.py`

# Activity on Loops

- In **pairs** or groups **up to three** work on the following activity.
- **loops\_ica.py**
- *Practice writing a solution that requires you to use a for-loop and a while-loop*

Remember to **check-in** with a TA before leaving class today!

In-Class “lab” Activity!

# Reminder: CS Laptop Loaner Program

---

- This course requires students to have a **laptop**
- I realize that not everybody might have one (nor necessarily need one for their desired major / path...)
- If you do not have a laptop for any reason... *not to worry!*
- The CS department's Systems staff has a notebook / laptop loaner program and will be able to loan you a notebook / laptop computer for the duration of the semester if you don't have one or if you cannot afford one.
  - Also available if your laptop is broken and under repair, we can arrange for you to receive a loaner laptop for a week or two until your own laptop is fixed

---

Interested? Link: [https://www.cs.virginia.edu/wiki/doku.php?id=cs\\_laptop\\_loaner](https://www.cs.virginia.edu/wiki/doku.php?id=cs_laptop_loaner)

*I am happy to be your sponsor. Please let me know.*



# Supplemental Slides

Looping through integers ~ some more information

Includes practice questions (and answers)

# Looping through integers - using the range() function

---

- `range(stop)`:
  - Gives all integers from 0 (inclusive) to stop (exclusive)
  - `range(5)` -> 0, 1, 2, 3, 4
- `range(start, stop)`:
  - Gives all integers from start (inclusive) to stop (exclusive)
  - `range(2,6)` -> 2, 3, 4, 5
  - start defaults to 0
- `range(start, stop, step)`:
  - Gives all integers from start (inclusive) to stop (exclusive), but it takes only every step item
  - The 3rd argument is the step-size, or increment size. It defaults to 1 (increase by 1).

**Note** that `range(start, stop)` and `range(start, stop, step)` behave similarly to string slicing

# An example of range(y, x, z)

---

- range(4, 8, 2)

- We start at 4,  $4 < 8$ , so we add it

# An example of range(y, x, z)

---

- range(4, 8, 2)
- 4
- We start at 4,  $4 < 8$ , so we add it

# An example of range(y, x, z)

---

- range(4, 8, 2)
- 4
- We start at 4,  $4 < 8$ , so we add it
- $4 + 2 = 6$ ,  $6 < 8$  so we add it



# An example of range(y, x, z)

---

- range(4, 8, 2)
- 4, 6
- We start at 4,  $4 < 8$ , so we add it
- $4 + 2 = 6$ ,  $6 < 8$  so we add it

# An example of range(y, x, z)

---

- range(4, 8, 2)
- 4, 6
- We start at 4,  $4 < 8$ , so we add it
- $4 + 2 = 6$ ,  $6 < 8$  so we add it
- $6 + 2 = 8$ , 8 is not  $< 8$  so we don't add it and stop (since the end is *exclusive*)

# An example of range(y, x, z)

---

- range(4, 8, 2)

- 4, 6

- We start at 4,  $4 < 8$ , so we add it

- $4 + 2 = 6$ ,  $6 < 8$  so we add it

- $6 + 2 = 8$ , 8 is not  $< 8$  so we don't add it and stop (since the end is *exclusive*)

A resource for **range()** can be found on the **docs.python.org** website:  
<https://docs.python.org/3/library/stdtypes.html#typeseq-range>

# Repetition using range()

---

```
for x in range(0, 5):  
    print(x)
```

- This repeats the code inside the **for** loop body **five times**:
  - The first time through the loop,  $x = 0$
  - The second time through the loop,  $x = 1$
  - The third time through the loop,  $x = 2$
  - The fourth time through the loop,  $x = 3$
  - The fifth time through the loop,  $x = 4$
- We do **\*not\*** repeat the loop when  $x = 5$

# More on range()

---

*# The range function works like this:*

```
range(start=0, stop, step=1)
```

```
for a in range(5): # 0,1,2,3,4
```

```
for b in range(1,5): # 1,2,3,4
```

```
for c in range(0,5,2) # 0,2,4
```

```
for d in range(10,-1,-1): # [10,9,8,7,6,5,4,3,2,1,0]
```

```
for e in range(-7): # []
```

# More on range()

---

*# count up*

```
for i in range(7): # calling range with one argument  
    print(i)
```

```
for i in range(1276, 8512): # range with two arguments  
    print(i)
```

# ★ Augmented Assignment Operators

Can be used to shorten the form of some basic math statements

- Only when the variable that is being assigned is also part of the expression on the right, i.e. -
  - **x = x + 1**
  - **num1 = num1 \* num2**
- The variable name does not need to be repeated if one of these operators is used -
  - **+=    -=    \*=    /=    %=    \*\*=    //=**
- Examples of use -
  - **a += 1    # a = a + 1**
  - **b -= 2    # b = b - 2**
  - **c \*= 5    # c = c \* 5**
  - **d /= 2    # d = d / 2**
- These operators are also sometimes called “update operators”
- Notice that the variable must have a value first before using it with one of these operators

# Q1: What numbers are printed?

---

```
for i in range(5):  
    print(i)
```



# Q1: What numbers are printed?

---

```
for i in range(5):  
    print(i)
```

```
0  
1  
2  
3  
4
```

## Q2: What numbers are printed?

---

```
for x in range(5, 2):  
    print(x)
```

## Q2: What numbers are printed?

---

```
for x in range(5, 2):  
    print(x)
```

<nothing>

**Why nothing?** The range starts at 5 and by default the increment is *\*positive\**. Therefore, there is no way to add something (positive) to 5 to get to something **LESS** than 5. Therefore, there is **nothing** in the range, and so the **for-loop** **doesn't iterate at all**.

*So be careful when creating your range!*

### Q3: What numbers are printed?

---

```
for j in range(2, 5):  
    print(j)
```

### Q3: What numbers are printed?

---

```
for j in range(2, 5):  
    print(j)
```

```
2  
3  
4
```

## Q4: What numbers are printed?

---

```
for i in range(6, 2, -2):  
    print(i)
```

## Q4: What numbers are printed?

---

```
for i in range(6, 2, -2):  
    print(i)
```

```
6  
4
```

Unlike Q2... The range starts at 6 and the increment is *\*negative\** (-2). Therefore, you can get from 6 to LESS than 6 by *subtracting*. Given 2 is exclusive, the range is only [6, 4] (not: [6, 4, 2]). So, the for-loop iterates two (2) times.  
*Sequence of numbers generated by range can positive or negative*

## Q5: What is the value of y?

---

```
y = 3
for x in range(3):
    y *= 2
print(y)
```



## Q5: What is the value of y?

---

```
y = 3
for x in range(3):
    y *= 2
print(y)
```

24

# Repetition with incrementing

---

```
total = 0
for count in range(5):
    total = total + count
print(total)
```

- This repeats the code inside the **for** loop body **five times**:
  - The first time through the loop, `count = 0`, `total = 0`
  - The second time through the loop, `count = 1`, `total = 1`
  - The third time through the loop, `count = 2`, `total = 3`
  - The fourth time through the loop, `count = 3`, `total = 6`
  - The fifth time through the loop, `count = 4`, `total = 10`
- We do **\*not\*** repeat the loop when `count = 5`