CS65: Introduction to Computer Science

For Loops (continued)
Nested For Loop
Sequence (Strings, List, and Tuple)

Md Alimoor Reza
Assistant Professor of Computer Science

Announcement

Lab #4 has been released on Friday (09/11/2025) and it will be due on October 16th Thursday for all sections

Content quiz#2 will take place on Friday (10/17/25) for section#1,#2 and Thursday (10/16/25) for section#3

- Topics:
  - Print formatting
  - `while` loop

Text Reading

https://python.swaroopch.com/control_flow.html

The For Loop

Review from Last Lecture

- More String Formatting
  - Hands-on Exercise
- For Loops
  - Hands on exercise 2,3,4: Describe code in English
- Function range()
  - Hands-on Exercise 5 (parts a-e)

Review: The for loop: a count-controlled loop

Count-Controlled Loop: iterates a specific number of times.

Use a **`for`** statement (a ***`for loop`***) to write count-controlled loops

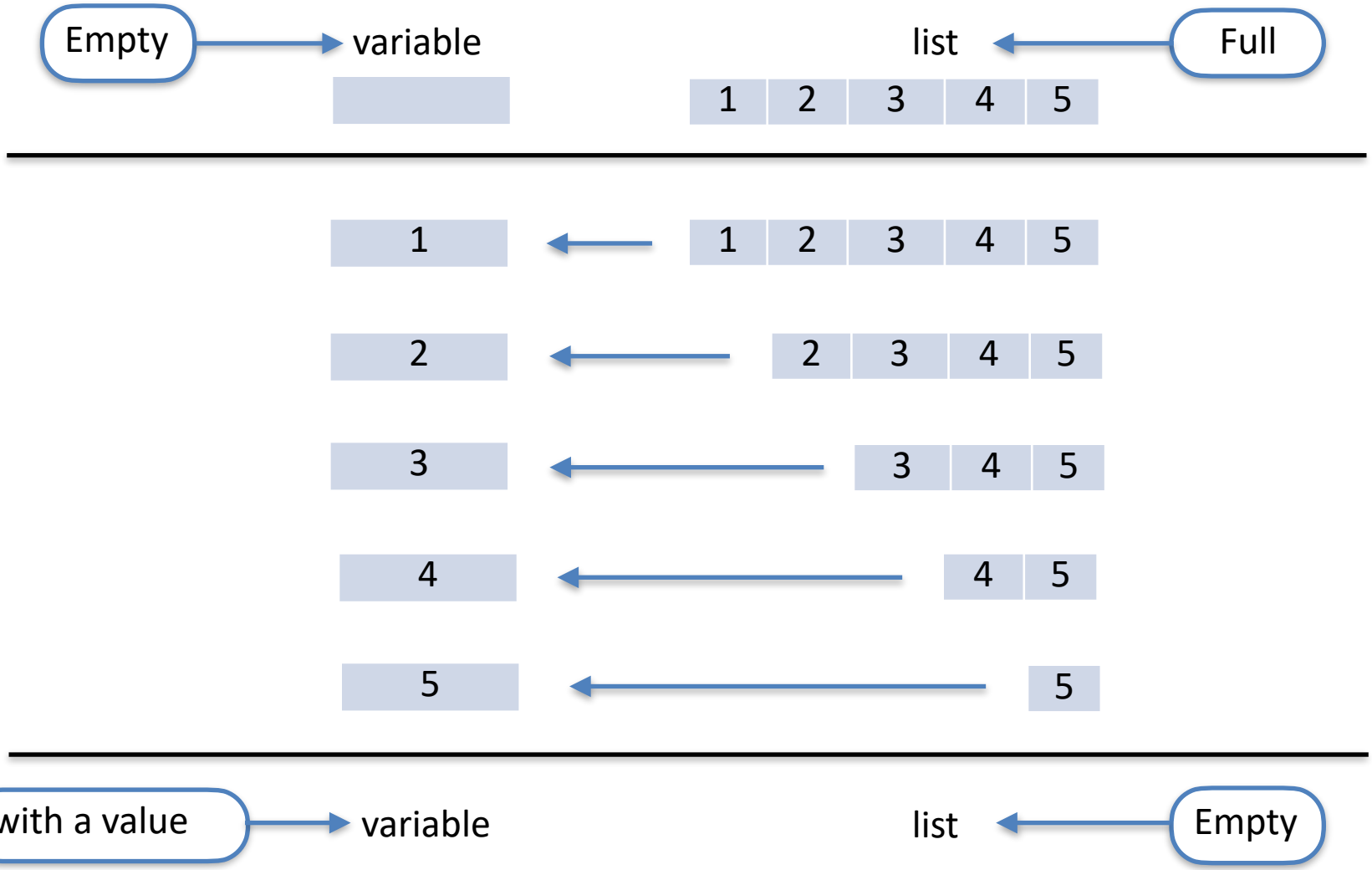
- Designed to work with a *sequence* of data items
  - Iterates once for each item in the sequence.
- General format:

```
for variable in [val1, val2, … , valn]:
      statements
```

Target variable: the variable which is the target of the assignment at the beginning of each iteration.

Review: **For** loop: concrete visualization

```
for variable in [1, 2, …, 5] :
    statements
```



Review: For Loop Examples

```
for name in ["Harry", "Ron", "Hermione"]:
    print(name)
```

```
Harry
Ron
Hermione
```

Review: Syntax of `for` loop vs Syntax of `while` loop

- **for** variable **in** [val_1, val_2, …, val_{15}] **:**
  statements

- Statements will be repeated sequentially from first to last item in a sequence

checking a condition

- **while condition expression :**

  statements

- **condition expression**: a boolean expression

- **statements** will repeatedly be executed until the **condition expression** becomes False

Review: `range()` function

The **`range`** function simplifies the process of writing a for loop

- `range` returns an *iterable* object
  - **Iterable**: contains a sequence of values that can be iterated (looped) over.

`range(5)` yields the same thing as [0,1,2,3,4]

Review: Examples

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

```
0
1
2
3
4
```

Review: Examples

```
for i in range(5):
    print("Hello World!")
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Review: Range()

If you pass two arguments to range, the first number is a starting value, and the second value is the ending limit.

`range(1,5)` yields the same things as [1,2,3,4]

Review: Range()

By default, the range function produces a sequence of numbers that increases by 1.

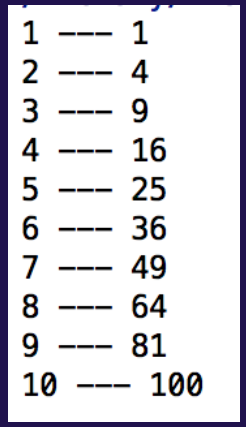
If you pass three arguments to range:

- The first number is the starting value
- The second number is the ending limit (not included)
- The third number is the step value (increment)

`range(1,10,2)` yields the same thing as [1,3,5,7,9]

Review: Exercise #4 – For Loops

a. Print out all integers between 1 and 200 (including 200)
b. Print out all even integers between 1 and 200 (including 200)
c. Print out all integers counting down from 50 to 1.
d. Print out all integers between 1 and 10, followed by the squares.



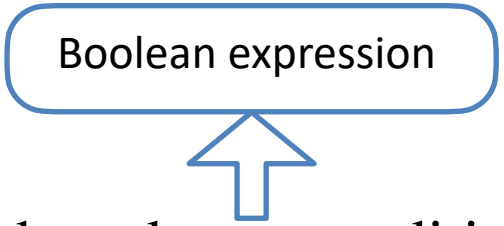
Today's Plan

- Value for loop vs. Index For Loop
  - Hands-on Exercise 5 (parts a-e)
- Nested For Loop
  - Hands on exercise
- Sequence
  - String
  - List
  - Tuple

When Should You Use **for** Loop vs. while Loop?

`for` loop

- use it when there is a **fixed** and **finite number of iterations**
  - "Do a calculation *10* or *N* times"
  - "Do a calculation from first to last item in a sequence"



`while` loop

- use it for **an indefinite number** of iterations based on a condition:
  - "Do until user enters END"
  - "Do until the number becomes negative"
  - "Do until we reach the end of the file with a special marker"

Value `for loop` vs Index `for loop`

common practice is
the index variables w

- So far we have seen the syntax of **value** `for loop`

```
for var in [10, 20, 30, 40, 50]:
    print(var)
```

- There is another form called **index** `for loop`

```
my_list = [10, 20, 30, 40, 50]
length = len(my_list)
for i in range(length):
    print( "location ", i, " value is ", my_list[i] )
```

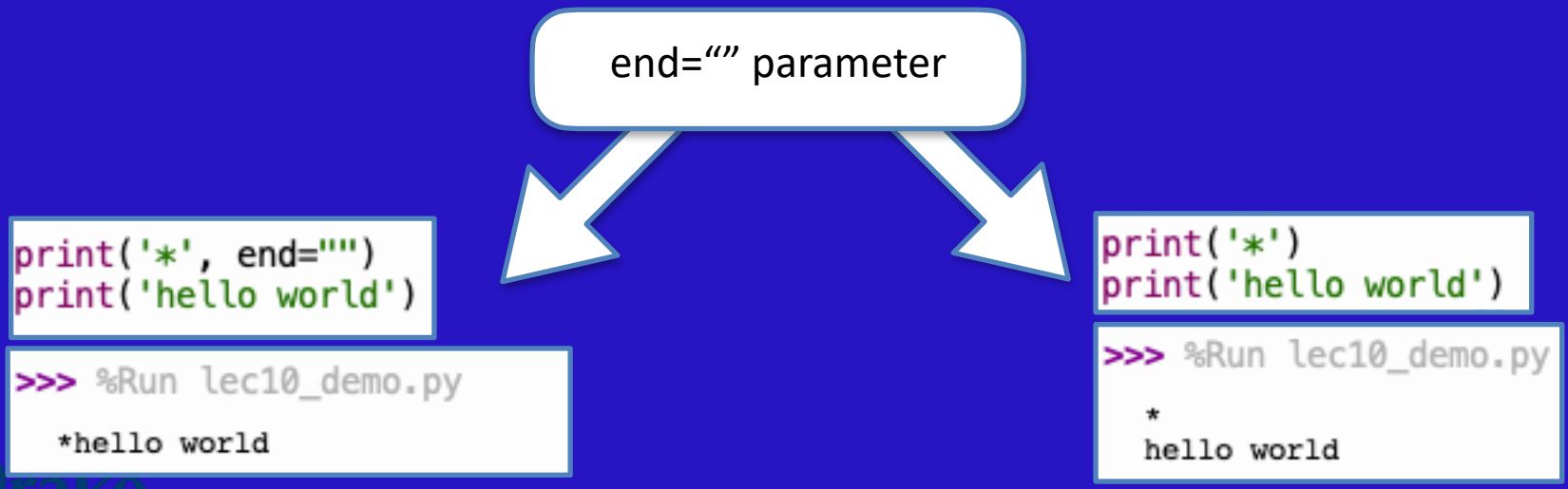
Value `for` loop vs Index `for` loop

- **value** `for loop`
  - directly assigns a value to the variable from the sequence
  - don't keep track of the indices
  - we have access to only value
    - **good**

- **index** `for loop`
  - generates all the indices of all elements in the list
  - each element can be accessed indirectly by that index
  - we have access to both *i) index* and *ii) value*
    - **better**

Exercises#1

- Write a for loop that will print '*' 5 times on the same line.
- Write a for loop that will print '*' 10 times on the same line.
- Write a for loop that will print '*' N times (prompt the user to enter this number) on the same line.



Exercise#2

- Write a code that will do the following:
  - Prompts the user for an integer number (between 1 to 100)
  - then **prints** all the odd numbers between 0 and the given number

- You have done it using while loop last time, now try it with for loop

Today's Plan

- Value for loop vs. Index For Loop
  - Hands-on Exercise 5 (parts a-e)
- Nested For Loop
  - Hands on exercise
- Sequence
  - String
  - List
  - Tuple

Nested **for** loops

- Putting one loop inside another
  - The first loop is called the outer loop
  - The second loop is called the inner loop

```
for i in range(3):
    # first line of outer loop
    for j in range(3):
        # first line of inner loop
        print("i: ", i, "j: ", j)
        # …
        # last line of inner loop, go back to beginning
    # …
    # last line of outer loop, go back to the beginning
```

Nested **for** loops

- Putting one loop inside another
  - The first loop is called the outer loop
  - The second loop is called the inner loop

- Here is simpler version:

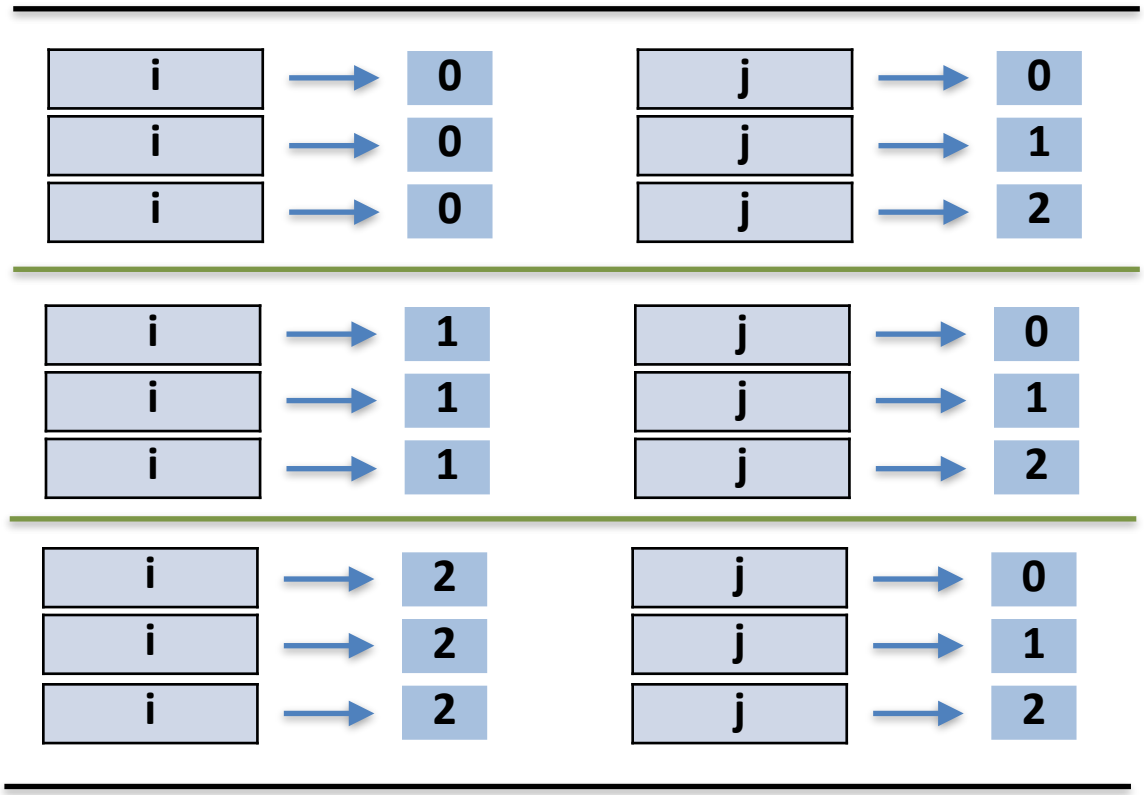
```
for i in range(3):
    for j in range(3):
        print("i: ", i, "j: ", j)
```

Visualization of nested **for** loop

```
# nested for loop
for i in range(3):
    print("Enters outer loop")
    for j in range(3):
        print("\tInner: i ->", i, " j ->", j)
```

Visualization of nested **for** loop

```
# nested for loop
for i in range(3):
    print("Enters outer loop")
    for j in range(3):
        print("\tInner: i ->", i, " j ->", j)
```



Thonny output: nested **for** loop

```
# nested for loop
for i in range(3):
    print("Enters outer loop")
    for j in range(3):
        print("\tInner: i ->", i, " j ->", j)
```

```
>>> %Run lec10_demo.py

  Enters outer loop
          Inner: i -> 0  j -> 0
          Inner: i -> 0  j -> 1
          Inner: i -> 0  j -> 2
  Enters outer loop
          Inner: i -> 1  j -> 0
          Inner: i -> 1  j -> 1
          Inner: i -> 1  j -> 2
  Enters outer loop
          Inner: i -> 2  j -> 0
          Inner: i -> 2  j -> 1
          Inner: i -> 2  j -> 2
```

Visualization of nested **for** loop

```
# nested for loop
for i in range(2):
    # first segment inside outer loop
    for j in range(3):
        # first segment inside inner loop
        print("i ->", i, " j ->", j)
        # next segment inside inner loop

        # ... segment insdie inner loop

    # next segment inside outer loop

    # ... segment inside outer loop
```

Notice the alignment (inner-loop)

Notice the alignment (outer-loop)

Exercise#3

- Build a left-facing-triangle in the shell output that looks like this:

```
>>> %Run left_facing_triangle.py

  *
  **
  ***
  ****
  *****
  ******
  *******
  ********
  *********
  **********
```

Side length = 10

```
>>> %Run left_facing_triangle.py

  *
  **
  ***
  ****
  *****
  ******
  *******
  ********
  *********
  **********
  ***********
  ************
  *************
  **************
  ***************
  ****************
  *****************
  ******************
  *******************
  ********************
```

Side length = 20

- You can use any special character of your choice as a brick, and my favorite is the '*' character :)

Exercise#3

- Build a right-facing-triangle in the shell output that looks like this:

```
>>> %Run right_facing_triangle.py
          *
         **
        ***
       ****
      *****
     ******
    *******
   ********
  *********
 **********
```

Side length = 10

```
>>> %Run right_facing_triangle.py
                    *
                   **
                  ***
                 ****
                *****
               ******
              *******
             ********
            *********
           **********
          ***********
         ************
        *************
       **************
      ***************
     ****************
    *****************
   ******************
  *******************
 ********************
```

Side length = 20

- You can use any special character of your choice as a brick, and my favorite is the '*' character :)

Today's Plan

- Value for loop vs. Index For Loop
  - Hands-on Exercise 5 (parts a-e)
- Nested For Loop
  - Hands on exercise
- Sequence
  - String
  - List
  - Tuple

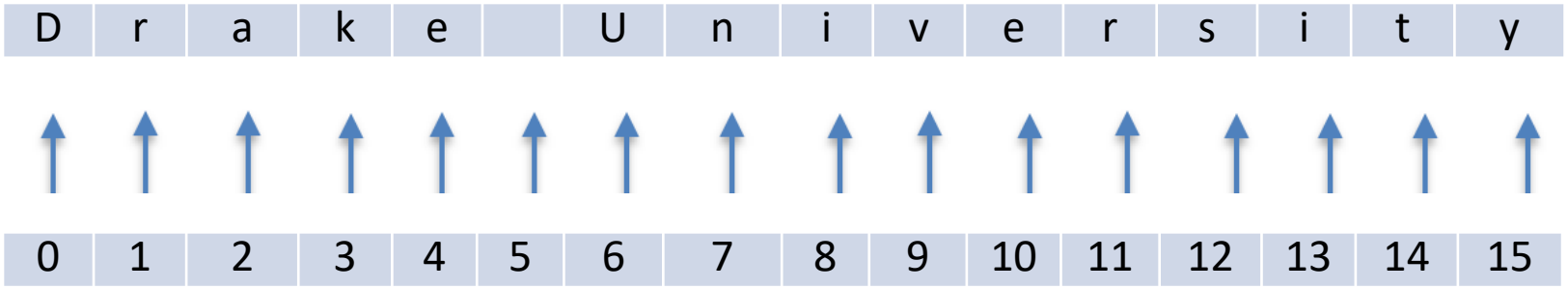
Sequence: Strings

- Sequence is an ordered group of elements (numbers, characters, etc)

- String is a sequence of characters
  - "Drake University"
  - "cs65:introduction_to_computer_science!"

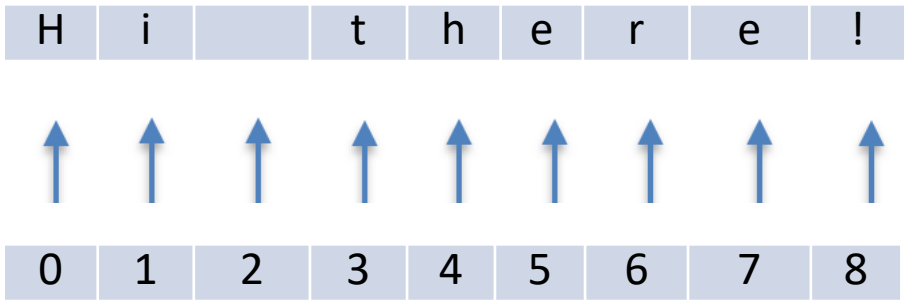
- *length* of a String is the total number of characters

- Each position in a sequence is marked with an **index** or **position**
  - Starts (from left) at position *0* and ends at position (*length-1*)
  - Start indexing from the *left* to *right*



Strings

- String is a sequence of characters
  - “”
  - “Hi there!”
- Each position is marked with an **index**
  - What are the *lengths* of the strings above?
  - Starts (from left) at position *0* and ends at position (*length-1*)



Strings

- String is a sequence of characters
  - "Drake University"
  - "cs65:introduction_to_computer_science!"

- Each position in a sequence is marked with an **index** or **position**
  - Starts (from left) at position *0* and ends at position (*length-1*)
  - Start indexing from the *left* to *right*
  - Python reports with an **IndexError** if the index goes out of bound

Length of a Sequence

- String is a sequence of characters
  - “”
  - “Hi there!”

- How can you find the length of a string?
  - Use built-in ***len()*** function

Demo: Length of a Sequence

- How can you find the length of a string?
  - Use built-in ***len()*** function

```
my_string1 = "hello@world"
my_string2 = "Hi there!"
my_string3 = ""

print("Length of \"hello@world\" is: ", len(my_string1))
print("Length of \"Hi there!\" is: ", len(my_string2))
print("Length of \"\" is: ", len(my_string3))
```

```
Shell
Python 3.7.9 (bundled)
>>> %cd /Users/reza/Class_and_Resea
    slides/lecture10
>>> %Run lec10_demo.py

  Length of "hello@world" is:  11
  Length of "Hi there!" is:  9
  Length of "" is:  0

>>>
```

Accessing Sequence Items with **Positive** Index
Left — — —> Right

- String is a sequence of characters

```
my_string1 = "Drake University"
```

- Access a specific item by appending ***brackets []*** containing an index

```
my_string1[0]     to access D
my_string1[1]     to access r
my_string1[2]     to access a
...
...
...
my_string1[15]    to access y
```

Accessing Sequence Items with **Negative** Index

Left <— — — Right

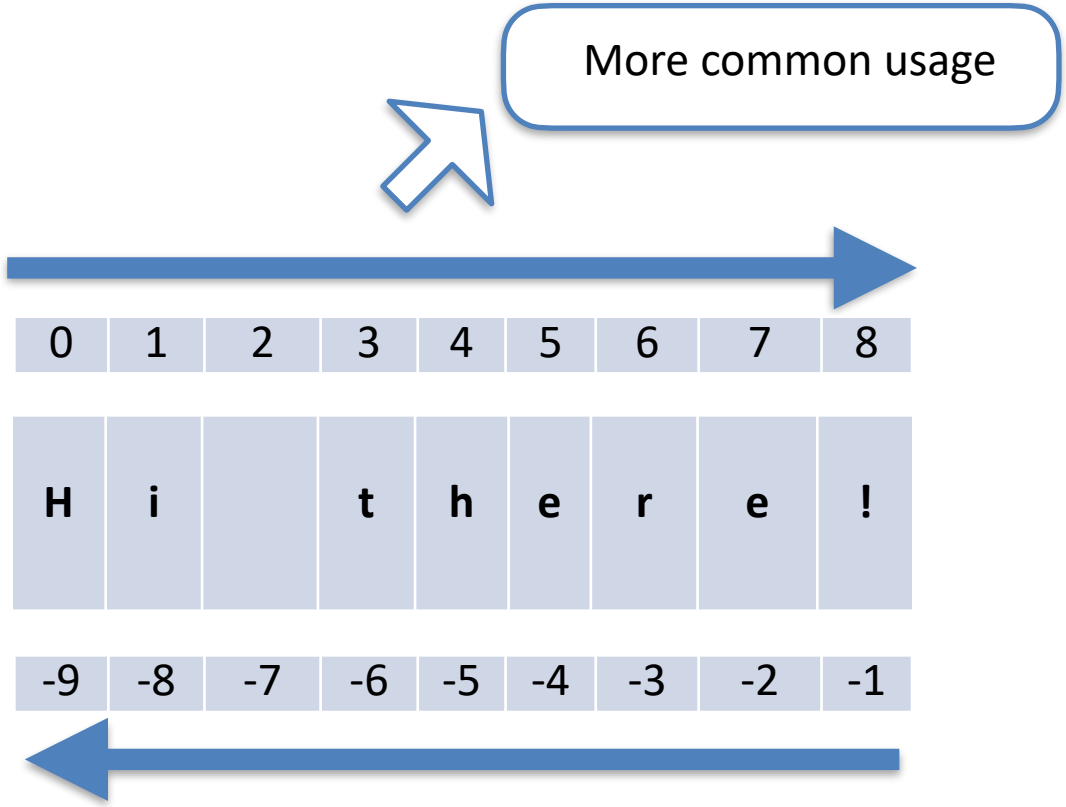
- String is a sequence of characters and negative indexing begins at the end with a **-1** (not zero anymore)

```
my_string1 = "Drake University"
```

- Access a specific item by appending ***brackets []*** containing an index

```
my_string1[-1]    to access  y
my_string1[-2]    to access  t
my_string1[-3]    to access  i
…
my_string1[-16]   to access  D
```

Summary of Indexing



Demo: Accessing Items with Index or Position

- How can you access an item in a sequence?
  - Use ***variable_name[index]***

```
15  # ----------------------------------------
16  # demo 2 accessing elements in a string
17  my_string1 = "Drake University"
18  my_string2 = "Hi there!"
19
20  vis = 1
21  if (vis):
22      print("Character at index = 0 is ", my_string1[0])
23      print("Character at index = 1 is ", my_string1[1])
24      print("Character at index = 2 is ", my_string1[2])
25      print("Character at index = 15 is ", my_string1[15])
26
27
```

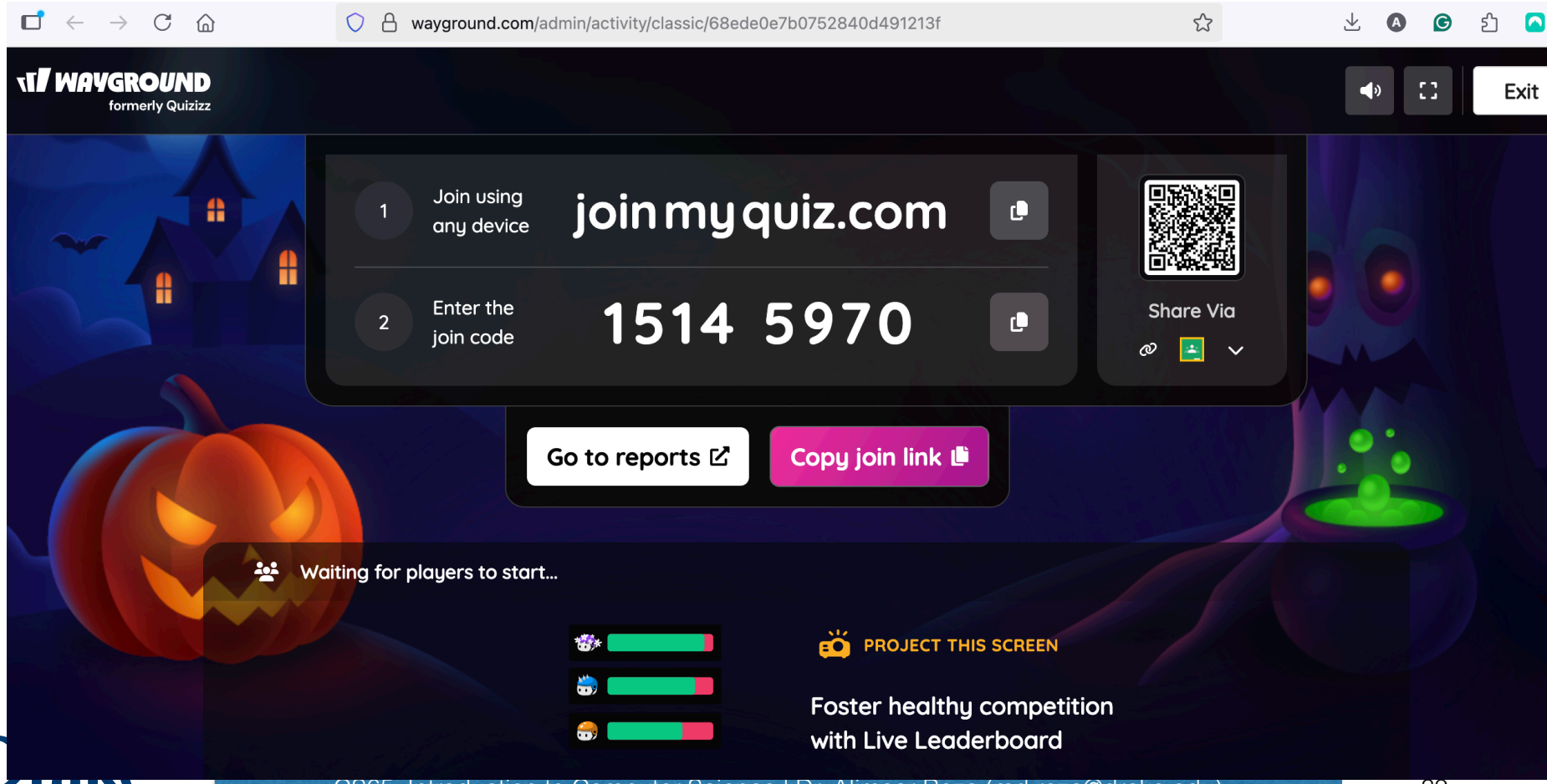
Shell

```
>>> %Run lec10_demo.py

Character at index = 0 is  D
Character at index = 1 is  r
Character at index = 2 is  a
Character at index = 15 is  y
```

Poll: String and index

- Please participate in the poll below
  - https://wayground.com/join?gc=15145970



Today's Plan

- Value for loop vs. Index For Loop
  - Hands-on Exercise 5 (parts a-e)
- Nested For Loop
  - Hands on exercise
- Sequence
  - String
  - List
  - Tuple

Demo: Length of a List

- How can you find the length of a string?
  - Use built-in ***len()*** function

my_list_demo2.py

```
1  my_list1 = ["Qingdao University", "hello", "world"]
2  my_list2 = [1, 2, 3, 4, 5]
3
4  print("lenth of my_list1 is: ", len(my_list1))
5  print("lenth of my_list2 is: ", len(my_list2))
6
7
```

Shell

```
>>> %Run my_list_demo2.py
  lenth of my_list1 is:  3
  lenth of my_list2 is:  5

>>>
```

Demo: Accessing Items with Index or Position

- How can you access an item in a List?
  - Use ***variable_name[index]***

my_list_demo1.py

```
1  my_list1 = ["Qingdao University", "hello", "world"]
2  my_list2 = [1, 2, 3, 4, 5]
3
4  print("Item at index = 0 is: ", my_list1[0])
5  print("Item at index = 0 is: ", my_list1[1])
6  print("Item at index = 0 is: ", my_list1[2])
```

Shell

```
 Item at index = 0 is:  world
>>> %Run my_list_demo1.py
 Item at index = 0 is:  Qingdao University
 Item at index = 0 is:  hello
 Item at index = 0 is:  world
```