

## more **for** loops.

**Week 4** | Lecture 2 (4.2.2)

if nothing else, write **#cleancode**

# Today's Content

- **Lecture 4.2.1**
  - **for loops**
  - **Reading:** 9.3, 9.4
- **Lecture 4.2.2**
  - **for loops on indices, nested loops**
  - **Reading:** 9.5 – 9.9

# for loops

- A **for** loop starts with the keyword **for**.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

# for loops

- Next, we provide the name of one or more variables.
- We have called the variable `character`, but you can call it whatever you like as long as it follows rules for naming a variable.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

```
for item1, item2 in iterable:  
    do something.
```

# for loops

- Our variable `character` will be bound to each of the items in the sequence in turn.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

# for loops

- Specify what the values are in.
- What is the iterable?
- An iterable is an object that can be iterated over.
- Strings are iterable (we know these from last week).
- Lists (**next week**) are iterable.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

# for loops

- As with the `while` loop, the `for` loop statement ends with a colon.
- This is how Python knows you are going to create a new block of code.

```
name = 'Sebastian'
```

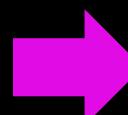
```
for character in name:  
    print(character)
```

# for loops

- Indenting four spaces tells Python what lines of code are in that block you want to repeated.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```



Indent

# Breakout Session 1

- We want to do some analysis of Dean Yip's Tweets.
- Before we can do this, we'll need to make the tweet all lower case and replace all the punctuations with white space.
- 'impact... Exciting' → 'impact exciting'



Chris Yip @UofTEngDean · Oct 23  
Replying to @UofTEngDean

great to hear from Prof. Bussman, Chair of @uoftmme about all the stuff that MechE do .... - amazing breadth of impact... exciting stuff across so many domains

Open your notebook

Click Link:

1. Breakout Session 1

# Today's Content

- Looping through indices with a **for** loop.
- Nested **for** loops.

# Looping Through Indices

- Last lecture we saw that we can use while loops to loop over the indices of a string.
- Then we saw that a for-loop requires less code but it iterates over the values, not the indices.

```
while
i = 0
while i < len(chrome_4):
    print(i, chrome_4[i])
    i += 1
```

```
for
for character in chrome_4:
    print(character)
```

# Looping Through Indices

- Can we use a for loop to loop over indices?

```
while
i = 0
while i < len(chrome_4):
    print(i, chrome_4[i])
    i += 1
```

```
for
for character in chrome_4:
    print(character)
```

# Looping on a `range()`

- Python has a built-in function called `range()` that can be used to generate a sequence of numbers. The general syntax of range is as follows:

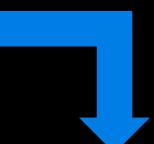
`range(start, stop, step)`

- Similar to the string slicing syntax:
  - The stop value is not included in the sequence of numbers generated.
  - Can omit start and step which will result in default values being used. `range(n) → range(0, n, 1)`

# Looping on a `range()`

- `range()` is typically used in a `for` loop to iterate over a sequence of numbers.
- `range()` is an iterable.

This thing has to be an iterable.



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

Open your  
notebook

Click Link:  
**2. Using `range()`**

## Example 1

- Add up all the even numbers between 1 and 100 using a **for** loop and `range()`.
- $2 + 4 + \dots + 96 + 98 + 100$

**Open your  
notebook**

**Click Link:**  
**3. Example 1**

# Breakout Session 2

- Write a function that returns the number of times that a character and the next character are the same.
- If you have a bug in a loop, with probability  $\sim 1$  its an off-by-one index error.

```
count_adjacent_repeats('abccdefffggh')
```

```
>>> 3
```

**Open your  
notebook**

**Click Link:**

**4. Breakout Session 2**

# Nested for Loops

```
for item in iterable:  
    do something.
```

- The bodies of loops can contain any statement, including other loops!
- When this occurs, it is known as a nested loop.

```
for i in range(10, 13):  
    for j in range(1, 5):  
        print(i, j)
```

Output

10, 1

10, 2

10, 3

...

# Nested for Loops

- The bodies of loops can contain any statement, including other loops!
- When this occurs, it is known as a nested loop.

**Open your notebook**

**Click Link:**  
**5. Nested for Loops**

# Turtles

- **Turtle** is a pre-installed Python library that enables users to create pictures and shapes by providing them with a virtual canvas.

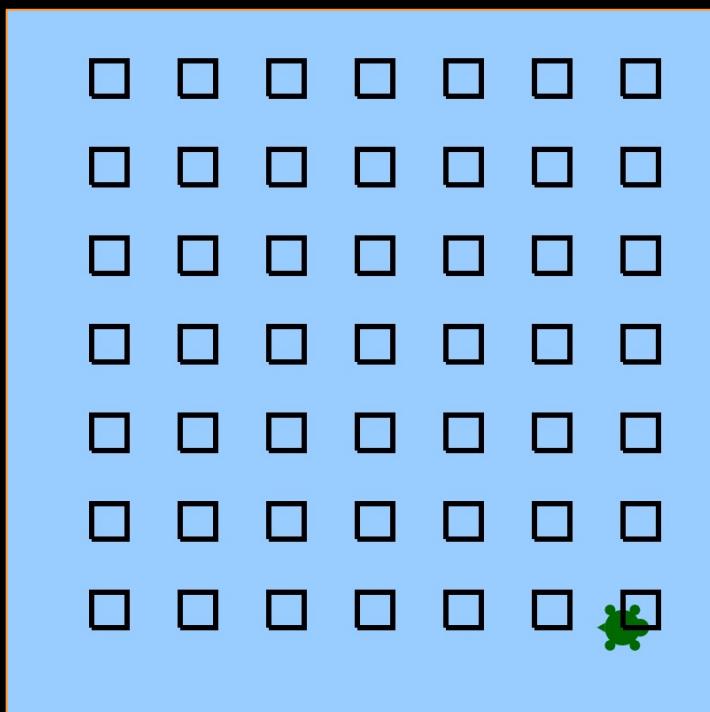


**Open your  
notebook**

**Click Link:  
6. Turtles**

# Draw A Grid

- Use Turtle and nested for loops to draw a grid.

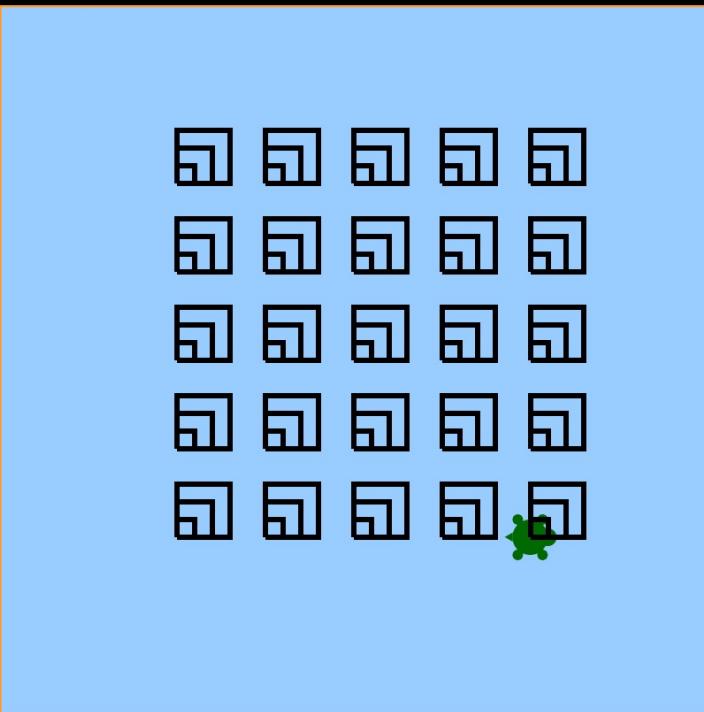


**Open your  
notebook**

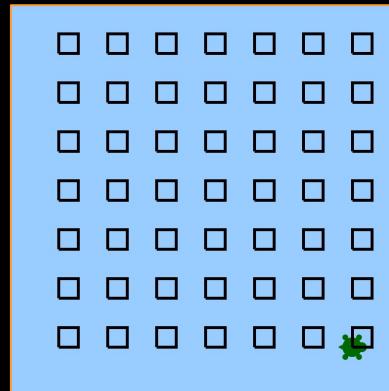
**Click Link:  
7. Grid**

# Breakout Session 3

- Use Turtle and nested for loops to draw the pattern below.



We just finished this.



**Open your  
notebook**

**Click Link:**  
**4. Breakout Session 3**

# Lecture Recap

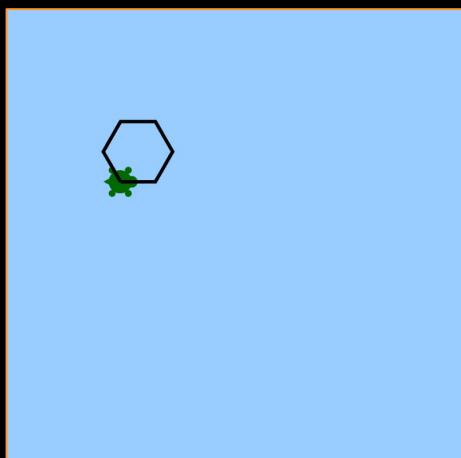
- The general form of a **for** loops.

```
for item in iterable:  
    do something.
```

- Iterable types have indices and items.
- For loops always iterate over the items in the iterable variable.
- Using **range(start, end, step)** we can keep track of where we are in a sequence (i.e. index).

# Advanced Turtles

- Create a function that draws a polygon at  $(x, y)$ .



```
draw_polygon(x, y,  
             num_sides,  
             side_length)
```

Open your  
notebook

Click Link:  
**8. Advanced Turtles**

## more **for** loops.

**Week 4** | Lecture 2 (4.2.2)

if nothing else, write **#cleancode**