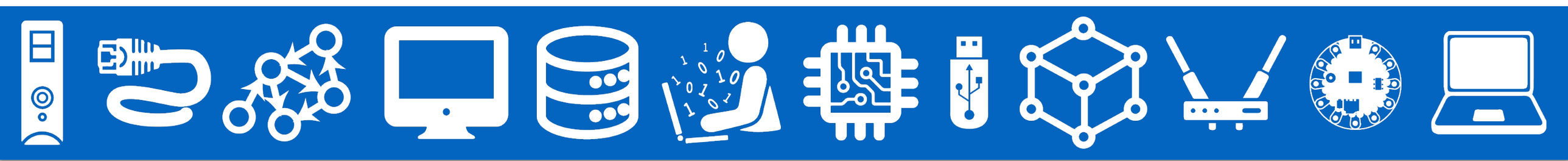
CS 134:
While Loops & Modules



Announcements & Logistics

- **HW 4** due Monday
- **Lab 4** (posted later today) will be a **Two-Part** Lab
  - Part 1 will be due next Wed/Thurs @10 pm
  - We will return feedback (including from tests not found in `runtests.py`) on Gradescope, but you will not be assigned a grade until Part 2
    - To receive credit for Part 1, YOU MUST SUBMIT a completed Part 1
    - You may fix any errors uncovered by the tests during your work on Part II
  - Part II will be due one week later than Part I
- **Colloquium Today**: @2:35pm in Wege Auditorium

Do You Have Any Questions?

Last Time

- Introduced nested lists
  - Lists can store any data type, including other lists
  - Bracket notation (e.g., `var[i]`) can be used to access successive levels of the nesting hierarchy

```
>>> lst[0]    # 0th element of lst
>>> lst[0][2] # 2th element of the 0th element of lst
```

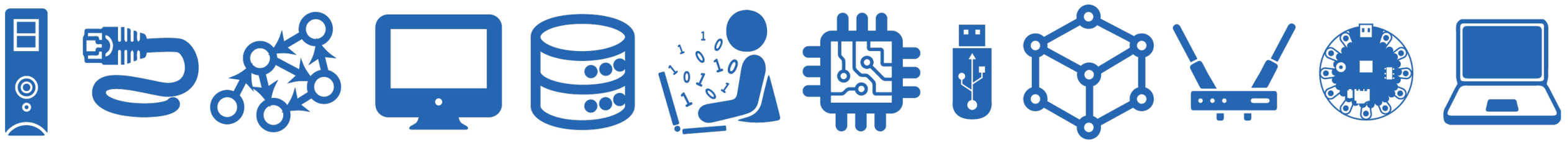
- Traversed sample nested lists using `for` loops

Today's Plan

- New iteration statement: the **`while`** loop
- Examples of iterating over nested sequences and collect/filter useful statistics

When you don't know when to stop (ahead of time):

While Loop



Story so far: `for` loops

- Python `for` loops are used to iterate over a **fixed sequence**
    - No need to know the sequence's length ahead of time
- Interpretation of for loops in Python:

```
for thing in things:
    # do something with thing
```

- Other programming languages (like Java) have `for` loops that require you to explicitly specify the length of the sequence or a stopping condition
- Thus Python for loops are sometimes called "**for each**" loops
- **Takeaway**: For loops in Python are meant to iterate directly over each item of a given ***iterable*** object (such as a sequence)

What if We Don't Know When to Stop?

- We always know the stopping condition of a `for` loop: when there are **no more elements in the sequence**

```
["A", "chilly", "autumn", "day"]
```

- Are there contexts where we don't know when to stop a loop?
    - Suppose you want to play a "guessing game" where a user repeatedly guesses numbers until they correctly guess the secret number

    - **How many times** should the loop execute?
    - **Under what condition** should the loop end?

The While Loop

A **`while`** loop executes the loop body 0 or more times, stopping once the loop condition evaluates to **`False`**:

```
while <boolean expression>:
    <loop body>
    <loop body>
    ...
```

Stopping condition

```
while False:
    print("never enters")
```

Loop body never executes

```
while True:
    print("never leaves")
```

"Infinite" loop!

While Loop Example

- Example of a **`while`** loop that depends on user input:

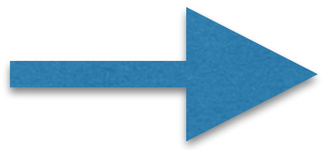
```
prompt = "Please enter a name (type quit to exit): "
name = input(prompt)
while (name != "quit"):
    print("Hi,", name)
    name = input(prompt)
print("Goodbye")
```

Stopping condition

While Loop Example: Print Halves

- Given a number, print all the positive "halves" : keep dividing **n** by **2** and printing the quotient until it becomes smaller than 0

```
def print_halves(n):
    while n > 0:
        print(n)
        n = n//2

print_halves(100)
```



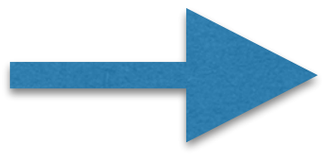
```
100
50
25
12
6
3
1
```

While Loop to Print Halves

- Given a number, print all the positive "halves" : keep dividing **n** by **2** and printing the quotient until it becomes smaller than 0

What does this do?

```
def print_halves(n):
    while n > 0:
        print(n)
        n = n//2

print_halves(100)
```



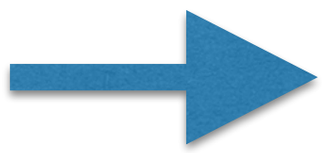
```
100
50
25
12
6
3
1
```

```
def print_halves2(n):
    while n > 0:
        print(n)
        n = n/2

print_halves2(100)
```

While Loop to Print Halves

- Given a number, print all the positive "halves" : keep dividing **n** by **2** and printing the quotient until it becomes smaller than 0

```
def print_halves(n):
    while n > 0:
        print(n)
        n = n//2

print_halves(100)
```



```
100
50
25
12
6
3
1
```

```
def print_halves2(n):
    while n > 0:
        print(n)
        n = n/2

print_halves2(100)
```

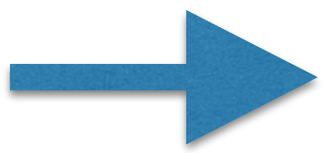
Float division!
Be careful!

While Loop to Print Halves

- Given a number, print all the positive "halves" : keep dividing **n** by 2 and printing the quotient until it becomes smaller than 0

What about this loop?

```
def print_halves(n):
    while n > 0:
        print(n)
        n = n//2

print_halves(100)
```



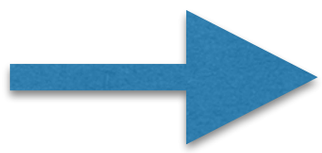
```
100
50
25
12
6
3
1
```

```
def print_halves3(n):
    while n > 0:
        print(n)
        n = n//2

print_halves3(100)
```

While Loop to Print Halves

- Given a number, print all the positive "halves" : keep dividing **n** by 2 and printing the quotient until it becomes smaller than 0

```
def print_halves(n):
    while n > 0:
        print(n)
        n = n//2

print_halves(100)
```



```
100
50
25
12
6
3
1
```

```
def print_halves3(n):
    while n > 0:
        print(n)
    n = n//2

print_halves3(100)
```

An infinite loop!
Indentation matters!

`while` and `if` side by side

```
if boolean_expresion:
    # statement 1
    # statement 2
    ....
    ....


# end of if
```

Execute body **once** if the **boolean expression** evaluates to true

```
while boolean_expresion:
    # statement 1
    # statement 2
    ....
    ....


# end of while
```

Keep executing body **as long as** the **boolean expression** (*continues*) to evaluate to true

Side by Side: for and while loops

Iteration steps are **implicit** in a **Python** for loop: `i` takes on values 0, 1, 2, 3, 4

```
for i in range(5):
    print('$' * i)
```

Explicitly **initialize** variable

Test stopping condition

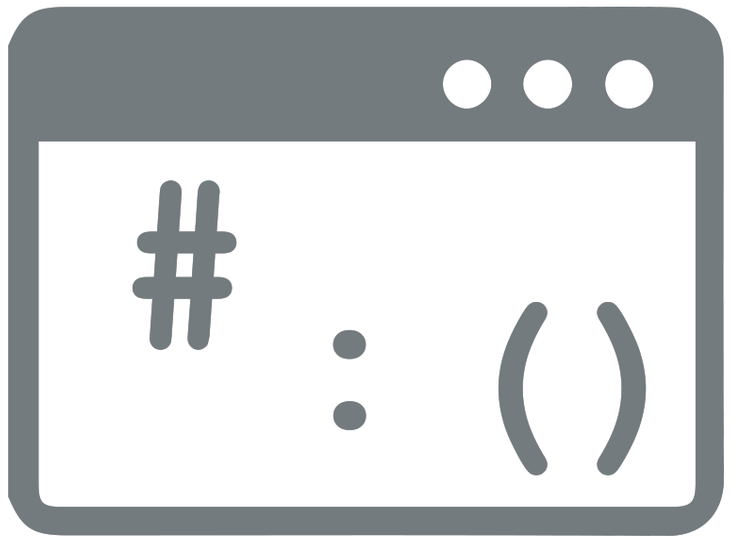
```
i = 0
while i < 5:
    print('$' * i)
    i = i + 1
```

Update value of variable used in test condition

Common while loops steps that we **explicitly** write:

- **Initialize** a variable used in the test condition
- **Test** condition that causes the loop to end when `False`
- Within the loop body, **update** the variable used in the test condition

While loop Examples



Let's Code!

Guessing Game

- Write a function that takes an integer "guess" as an argument, and repeatedly prompts the user to guess a number until they guess correctly.
  - To help, your code should tell the user "guess higher" or "guess lower" after each incorrect guess.
  - When the correct number is guessed, the function should print "Correct guess!" and then return.

Summary Stats

- Function 1: Count subject
  - Given a course subject (string), count the number of times this subject was taken in the given schedule (list of list of strings)
- Function 1: Most popular subject
  - Given a list of course subjects (list of strings) and a course schedule (list of lists of strings), return a list of course subjects that were taken the most times throughout the course schedule

The end!