Announcements

• A3 is being graded this week
• A4 is out. Due next Thursday.
• Prelim 2
  ▪ Tuesday, April 24th, 7:30-9:00pm
  ▪ Please go to the same room you went for Prelim 1
  ▪ Conflict assignment on CMS, due 11:59pm Tonight
Recall: For Loops

```python
for x in grades:
    print(x)
```

- **loop sequence**: `grades`
- **loop variable**: `x`
- **body**: `print(x)`

To execute the for-loop:

1. Check if there is a “next” element of **loop sequence**
2. If so:
   - `assign` next sequence element to **loop variable**
   - Execute all of **the body**
   - Go back to Line 1
3. If not, terminate execution.
Different types of Repetition

1. Process each item in a sequence
   - Compute statistics for a dataset.
   - Send all your contacts an email.

2. Do something $n$ times
   - Draw a checkers board.
   - Run a protein-folding simulation for $10^6$ time steps.

3. Do something an unknown number of times
   - Fly up until you’re near the ceiling.
   - Play hangman until 6 strikes.
Beyond Sequences: The **while-loop**

\[ \textbf{while} \ <\textit{condition}>:\]

\[
\begin{align*}
\text{statement 1} \\
\ldots \ \\
\text{statement n}
\end{align*}
\]

- **Relationship to for-loop**
  - Broader notion of “keep working until done”
  - Must explicitly ensure condition becomes false
  - *You* explicitly manage what changes per iteration
import random

num = random.randint(0, 10)
guessed_it = False
print('I’m thinking of a number.

while not guessed_it:
    guess = int(input('Guess it: '))
guessed_it = (num == guess)
print('Well done!')
Q1: What gets printed?

\[
\begin{align*}
\text{a} &= 0 \\
\text{while } \text{a} < 1: & \quad \text{a} = \text{a} + 1 \\
\text{print(a)} & \\
\end{align*}
\]

\[
\begin{align*}
\text{a} &= 0 \\
\text{while } \text{a} < 2: & \quad \text{a} = \text{a} + 1 \\
\text{print(a)} & \\
\end{align*}
\]

\[
\begin{align*}
\text{a} &= 0 \\
\text{while } \text{a} > 2: & \quad \text{a} = \text{a} + 1 \\
\text{print(a)} & \\
\end{align*}
\]
A1: What gets printed?

\[
\begin{align*}
& a = 0 \\
& \text{while } a < 1: \\
& \quad a = a + 1 \\
& \text{print}(a)
\end{align*}
\]

\[
\begin{align*}
& a = 0 \\
& \text{while } a < 2: \\
& \quad a = a + 1 \\
& \text{print}(a)
\end{align*}
\]

\[
\begin{align*}
& a = 0 \\
& \text{while } a > 2: \\
& \quad a = a + 1 \\
& \text{print}(a)
\end{align*}
\]

\[
\begin{align*}
& 1 \\
& 2 \\
& 0
\end{align*}
\]
Q2: What gets printed?

a = 4
while a > 0:
    a = a - 1

print(a)

a = 0
while a < 3:
    if a < 2:
        a = a + 1

print(a)
A2: What gets printed?

```
a = 4
while a > 0:
    a = a - 1
print(a)
```

```
a = 0
while a < 3:
    if a < 2:
        a = a + 1
print(a)
```

Infinite loop!
Q3: What gets printed?

```
a = 8
b = 12

while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a

print(a)
```

A: Infinite Loop!
B: 8
C: 12
D: 4
E: I don’t know
A3: What gets printed?

\[ a = 8 \]
\[ b = 12 \]

\[ \text{while } a \neq b: \]
\[ \quad \text{if } a > b: \]
\[ \quad \quad a = a - b \]
\[ \quad \text{else:} \]
\[ \quad \quad b = b - a \]

print(a)

A: Infinite Loop!
B: 8
C: 12
D: 4  CORRECT
E: I don’t know

This is Euclid’s Algorithm for finding the greatest common factor of two positive integers.

Trivia: It is one of the oldest recorded algorithms (~300 B.C.)
for vs. while

- You can almost always use either
- Sometimes **for** is better
- Sometimes **while** is better
for vs. while

do something n times

for k in range(n):
    # do something

k = 0
while k < n:
    # do something
    k = k+1

Must remember to increment

My preference? for-loop
for vs. while

do something an unknown number of times

for \textbf{k} in range(BIG\_NUM):
    \# do something
    if time to stop:
        break

while \textbf{not time to stop}:
    \# do something

My preference? while-loop
**for vs. while**

**do something to each element of a sequence**

```python
for k in range(len(seq)):
    seq[k] = seq[k] + 1
```

```python
k = 0
while k < len(seq):
    seq[k] = seq[k] + 1
    k = k + 1
```

while is more flexible, but often requires more code

My preference? for-loop
for vs. while

do something until a limit is reached
make a table of squares up to N

```python
seq = []
n = math.floor(sqrt(N)) + 1
for k in range(n):
    seq.append(k*k)
```

for-loop requires you to know how many iterations you want **ahead of time**

```python
seq = []
k = 0
while k*k < N:
    seq.append(k*k)
k = k+1
```
can use complex expressions to check if a task is done

My preference? while-loop
for vs. while

change a sequence’s length
remove all 3’s for list nums

for i in list(range(len(nums))):
    if nums[i] == 3:
        del num[i]

while 3 in nums:
    nums.remove(3)

IndexError: list index out of range

is this not beautiful?

My preference? while-loop
for vs. while

Fibonacci numbers:
- $F_0 = 1$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$

find 1\textsuperscript{st} n Fibonacci numbers

fib = [1, 1]
for k in range(2,n):
    fib.append(fib[-1] + fib[-2])

fib = [1, 1]
while len(fib) < n:
    fib.append(fib[-1] + fib[-2])

loop variable not always \textbf{used}

loop variable not always \textbf{needed} at all

My preference? while-loop
import random, hangman

word_list = [ ... words we want user to guess .. ]
N_GUESSES = 10
secret = hangman.SecretWord(random.choice(word_list))

for n in list(range(N_GUESSES)):
    secret.word_so_far()
    user_guess = input("Guess a letter: ")
    secret.apply_guess(user_guess):
        if secret.is_solved():
            print("YOU WIN!!!")
            break  # jumps out of the for-loop
    secret.reveal()
Improving Hangman with while

import random, hangman
word_list = [ ... words we want user to guess .. ]
N_GUESSES = 10  MAX_STRIKES = 10
secret = hangman.SecretWord(random.choice(word_list))
n_strikes = 0
for n in list(range(N_GUESSES)):
    secret.word_so_far()
    user_guess = input("Guess a letter: ")
    bad_guess =
    secret.apply_guess(user_guess)
    if secret.is_solved():
        print("YOU WIN!!!")
        break  #jumps out of the loop
    if bad_guess:
        n_strikes = n_strikes + 1
secret.reveal()
Using **while-loops** Instead of **for-loops**

### Advantages

- Better for **modifying data**
  - More natural than range
  - Works better with deletion
- Better for **convergent tasks**
  - Loop until calculation done
  - Exact steps are unknown
- Easier to **stop early**
  - Just set loop var to False

### Disadvantages

- **Infinite loops** more likely
  - Easy to forget loop vars
  - Or get stop condition wrong
- **Require** more management
  - Initialize the condition?
  - Update the condition?