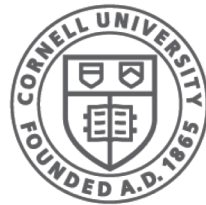


<http://www.cs.cornell.edu/courses/cs1110/2019sp>

Lecture 21: While Loops (Sections 7.3, 7.4)

CS 1110

Introduction to Computing Using Python



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

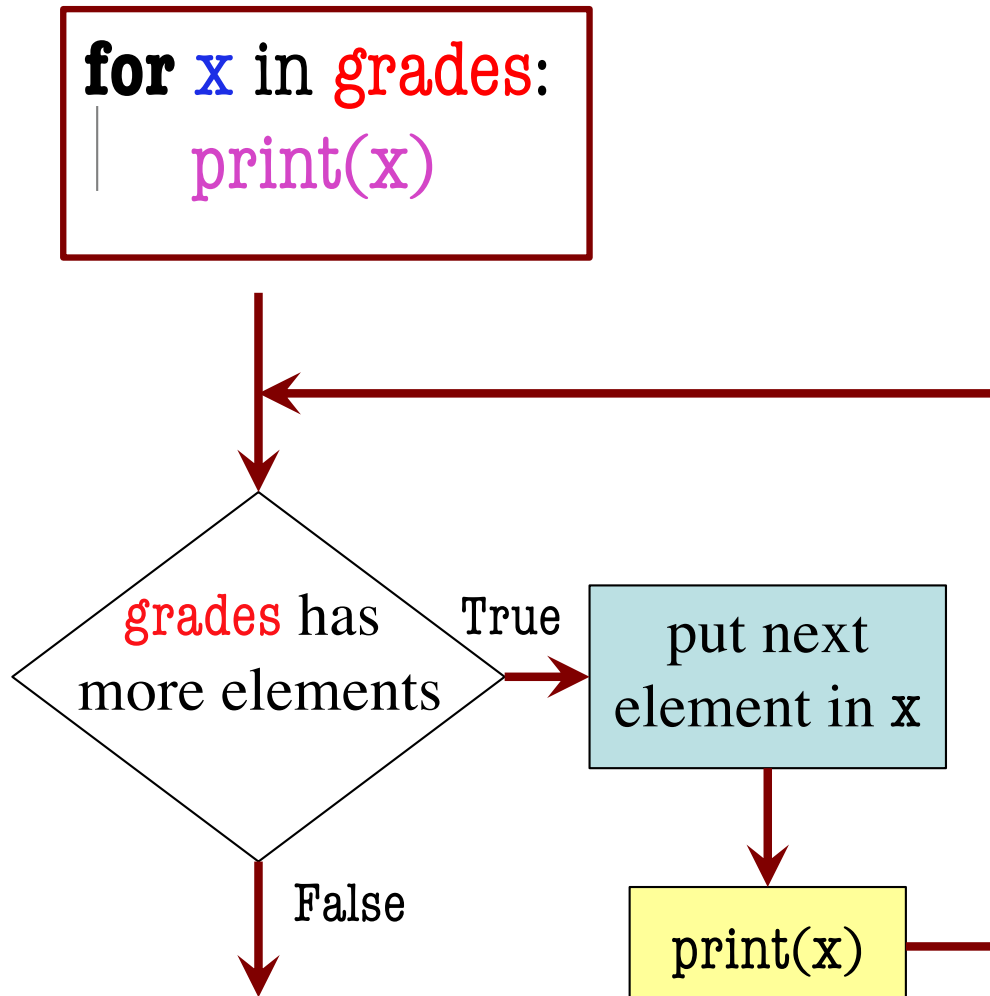
[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Recall: For Loops

- **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 execution₂



Different types of Repetition

1. Process each item in a sequence

- Compute statistics for a dataset.
- Send all your contacts an email.

```
for x in sequence:  
    process x
```

2. Do something n times

- Draw a checkers board.
- Run a protein-folding simulation for 10^6 time steps.

```
for x in range(n):  
    do something
```

3. Do something an unknown number of times

- Fly up until you're near the ceiling.
- Play hangman until 6 strikes.



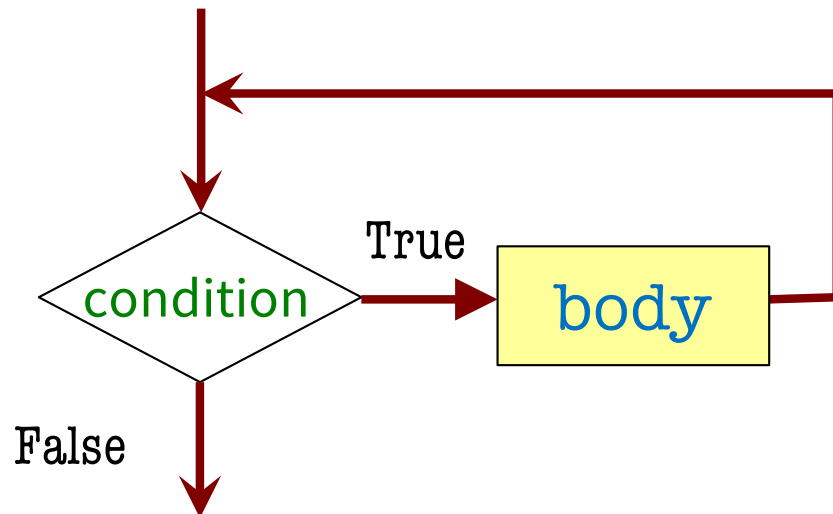
???

Beyond Sequences: The **while**-loop

while *<condition>*:

```
statement 1  
...  
statement n
```

} body



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

While-Loops and Flow

```
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!')
```

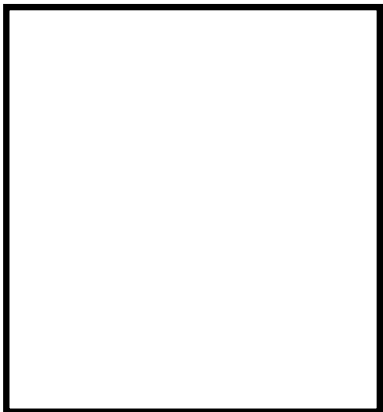
I'm thinking of a number.
Guess it: 6
Guess it: 2
Guess it: 1
Guess it: 4
Well done!

Q1: What gets printed?

A: 0 B: 1 C: 2 D: 3
E: Infinite Loop!

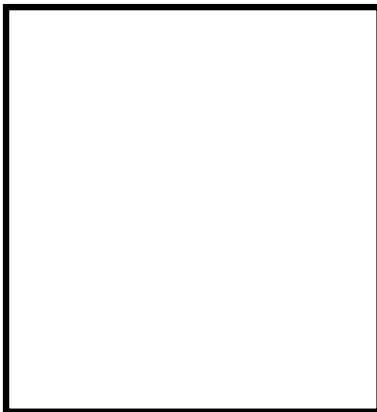
```
a = 0
while a < 1:
    a = a + 1
```

```
print(a)
```



```
a = 0
while a < 2:
    a = a + 1
```

```
print(a)
```



```
a = 0
while a > 2:
    a = a + 1
```

```
print(a)
```



Q2: What gets printed?

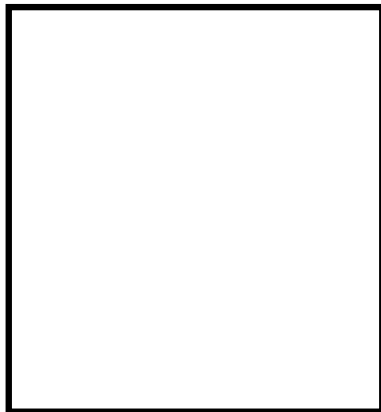
A: 0 B: 1 C: 2 D: 3
E: Infinite Loop!

```
a = 4
```

```
while a > 0:
```

```
    a = a - 1
```

```
print(a)
```



```
a = 0
```

```
while a < 3:
```

```
    if a < 2:
```

```
        a = a + 1
```

```
print(a)
```



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

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 k in range(BIG_??_NUM):  
    # do something  
    if time to stop:  
        break
```

```
while not time to stop:  
    # do something
```

My preference? while-loop

for vs. while

do something to each element of a sequence

for `k in range(len(seq)):`

| `seq[k] = seq[k]+1`

`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

```
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**

```
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 range(len(nums)):
    if nums[i] == 3:
        del num[i]
```

IndexError: list index out of range

```
while 3 in nums:
    nums.remove(3)
```

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 1st 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 **used**

loop variable not
always **needed** at all

My preference? while-loop

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?