Iterative Execution

- For statements loop over a sequence of things
  - Data

```
for name in [“Bob”, “Alice”, “Sue”]:
```

  - Counting

```
for i in [10,11,12,13,14]:
```

- The range function makes counting convenient
  - Returns each number from start to end

```
for i in range(10,15):
```

  - Third argument is *step*, allowing counting by 2, backwards, etc.

```
for i in range(10,0,-2):
```

- These are all definite loops
  - We know before the loop starts how many times it needs to repeat

Indefinite Loops

- An indefinite loop we don’t know when it will stop up front
  - Repeat until it starts to rain
  - Repeat until I say stop
- Typical use cases
  - Repeat until the user presses a certain key
    - e.g. would you like to play again?
  - Repeat until a certain variable reaches some value
    - e.g. repeat until someone wins

Indefinite Loops

- The *while* statement

```
while condition
    statement
```

- Same structure as an if statement
  - *if*: execute this block of code once, if condition is true
  - *while*: execute this block of code over and over, *as long as* condition is true

- Same rules for conditions in an if statement
  - Any expression that evaluates to a Boolean
  - But! Condition must change, or the while loop will never end!

Indefinite Loop Examples

- Counting loop

```
i = 0
while i < 10:
  print(i)
  i = i + 1
```

- User interaction loop
  - Keeps asking until the user types something (not blank)

```
name = ‘’
while name == ‘’:
  name = input(“Please enter your name: ”)
```

Exercise

- Quick sanity check!
- What will the following loops print to the screen?

```
i = 7
while i > 4:
  print(i)
  i = i - 1

for x in range(0,3):
  y = x + 1
  msg = ‘’
  while y < 4:
    msg = msg + str(y)
    y = y + 1
  print(msg)
```

Breaking Out

- Sometimes you want to end a loop early
  - Applies to both *for* and *while* loops
  - The *break* statement immediately ends the innermost loop
- This example will only print 0,1,2 and 3

```
for i in range(0,5):
  print(i)
  if i == 3:
    break
```

Breaking Out

- The continue statement is similar
  - Instead of exiting the loop, goes immediately to the next iteration (back to the top)
- This example will print "AHHHHHHHH" only after the odd numbers:

```
for i in range(0,5):
  print(i)
  if i % 2 == 0:
    continue
  print("AHHHHHHHH")
```