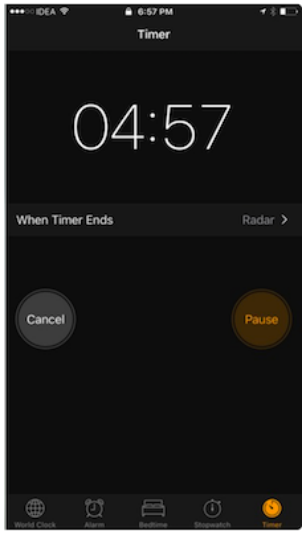
Iteration – While Loops

CS111 Computer Programming

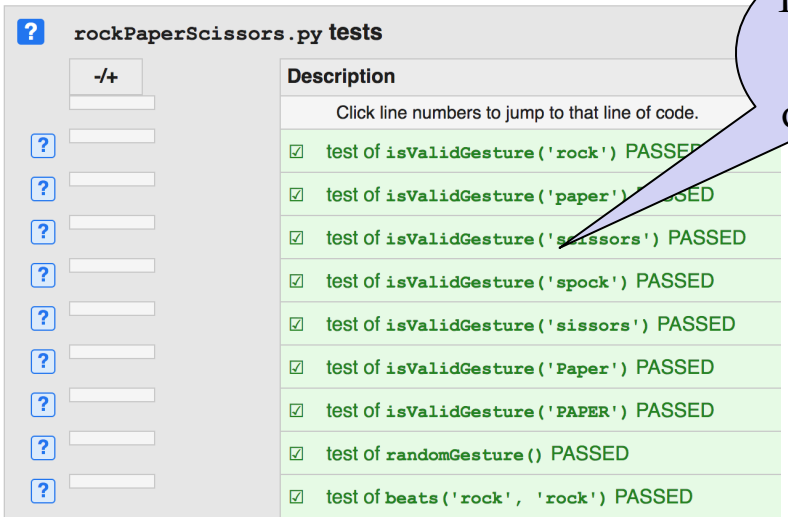
Department of Computer Science
Wellesley College

Motivation for iteration

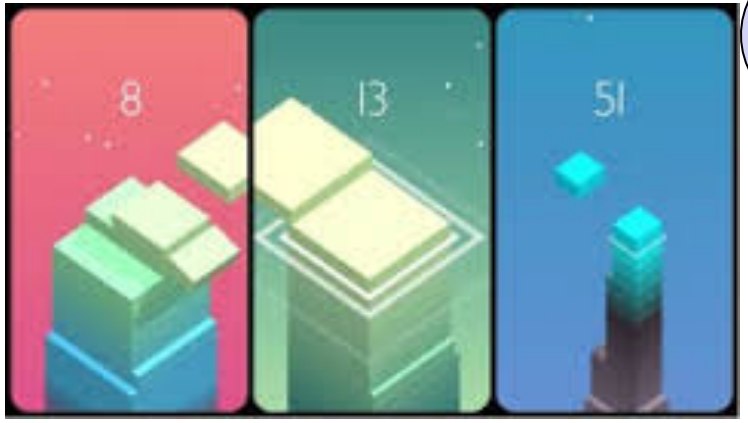
Concepts in this slide: Iteration is a problem-solving strategy found in many situations.



Display time until no more time left



Keep coding until all test cases passed



Play until blocks too small to stack

What is Iteration?

Concepts in this slide: Definition of iteration; stopping condition.

Repeated execution of a set of statements

Keep repeating….

until **stopping** condition is reached

The `while` loops

Concepts in this slide: Syntax of `while` loops.

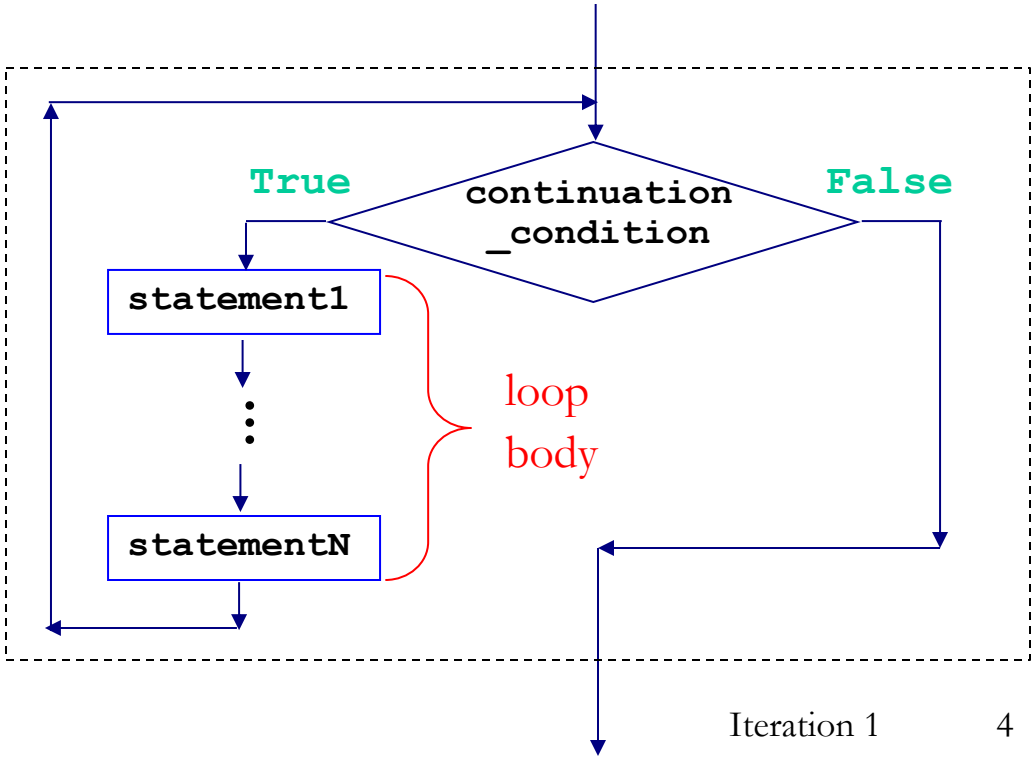
`while` loops are a fundamental mechanism for expressing iteration

keyword indicating while loop

a boolean expression denoting whether to iterate through the body of the loop one more time.

```
while continuation_condition :
    statement1
        ⋮
    statementN
```

body of loop = actions to perform if the continuation condition is true



Concepts in this slide: Code for a while loop with flexible stopping.

`while` loops and user input

```
name = input('Please enter your name: ')
while (name.lower() != 'quit'):
      print('Hi,', name)
      name = input('Please enter your name: ')
print('Goodbye')
```

```
Please enter your name: Ted
Hi, Ted
Please enter your name: Marshall
Hi, Marshall
Please enter your name: Lily
Hi, Lily
Please enter your name: quit
Goodbye
```

`while` loops are not just for user input.

Useful for other problems too.

`while` Loop Example: printHalves


```
def printHalves(n):
    '''Prints positive successive halves of n'''
    while n > 0:
        print(n)
        n = n//2
```

```
In[2]: printHalves(22)
```

Execute this function on paper and fill in the results.

A slight variation of `printHalves`:

Concepts in this slide: How does an infinite loop happens?

```
def printHalves2(n):
    '''Attempts to print positive successive
    halves of n'''
    while n > 0:
        print(n)
    n = n//2
```

What's the output? `printHalves2(22)`

```
In[2]: printHalves2(22)
22
22
22
22
22
22
22
22
...
```

An "infinite loop"
(in Thonny, stop with
Ctrl-C Ctrl-C)

Accumulating a result with a `while` loop

Concepts in this slide: The recipe for implementing the accumulating pattern.

It is common to use a **`while`** loop with "accumulators" that accumulate results from processing the elements.

Below is defined the **`sumHalves`** function that takes an nonnegative integer and returns the sum of the values printed by **`printHalves`** (slide 6).

```
In [3]: sumHalves(22)
Out[3]: 41 # 22 + 11 + 5 + 2 + 1
```

```
def sumHalves(n):
    sumSoFar = 0
    while n > 0:
        sumSoFar = sumSoFar + n # or sumSoFar += n
        n = n//2
    return sumSoFar
```

initialize accumulator

update accumulator

return accumulator

Iteration Tables [Model of execution]

An iteration is a step-by-step process characterized by a collection of **state variables** that determine the next step of the process from the current one. E.g the state variables of **`sumHalves`** are **`n`** and **`sumSoFar`**.

The execution of an iteration can be summarized by an **iteration table**, where columns are labeled by state variables and each row represents the values of the state variables at one point in time.

Example: iteration table for **`sumHalves(22)`**:

step is not a state variable but a label that allows us to distinguish rows

step	**n**	**sumSoFar**
0	22	0
1	11	22
2	5	33
3	2	38
4	1	40
5	0	41

Iteration Rules

Concepts in this slide: The steps for completing an iteration table.

An iteration is governed by

- **initializing the state variables** to appropriate values;
- specifying **iteration rules** for how the next row of the iteration table is determined from the previous one;
- specifying the **continuation condition** (alternatively, stopping condition)

step	**n**	**sumSoFar**
0	22	0
1	11	22
2	5	33
3	2	38
4	1	40
5	0	41

initial values of state variables

continue while n > 0 (stop when n <= 0)

Iteration rules for **`sumHalves`**:

- next **`sumSoFar`** is current **`sumSoFar`** plus current **`n`**.
- next **`n`** is current **`n`** divided by 2.

Printing the iteration table in a loop

Concepts in this slide: Using print statements to understand variable states.

By adding one **print** statement **right before** the loop and another print statement as the last statement in loop body, you can print each row of the iteration table.

```
def sumHalvesTable(n):
    sumSoFar = 0
    print 'n:', n, '| sumSoFar:', sumSoFar
    while n > 0:
        sumSoFar = sumSoFar + n # or sumSoFar += n
        n = n//2
        print 'n:', n, '| sumSoFar:', sumSoFar
    return sumSoFar
```

```
In[4]: sumHalvesTable(22)
n: 22 | sumSoFar: 0
n: 11 | sumSoFar: 22
n: 5 | sumSoFar: 33
n: 2 | sumSoFar: 38
n: 1 | sumSoFar: 40
n: 0 | sumSoFar: 41
Out[17]: 41
```

What is the result? Fill in the table.


```
def sumHalves2(n):
    '''Prints positive successive halves of n'''
    sumSoFar = 0
    while n > 0:
        n = n//2
        sumSoFar = sumSoFar + n
    return sumSoFar

sumHalves2(22)
```

Compare this function definition to that of **sumHalves** in slide 10. How do the two definitions differ?

step	n	sumSoFar
0	22	0
1	11	
2	5	
3	2	
4	1	
5	0	

Premature return done wrong

GOTCHA!

```
def sumHalvesBroken(n):
    '''Broken version of returns sum of halves of n'''
    sumSoFar = 0
    while n > 0:
        sumSoFar = sumSoFar + n # or sumSoFar += n
        n = n//2
        return sumSoFar # wrong indentation!
                        # exits function after first
                        # loop iteration. Sometimes we
                        # want this, but not here!
```

Wrong indentation within the loop.
Function returns after first iteration

```
In [4]: sumHalvesBroken(22)
Out[4]: 22
```

Test your knowledge

1. Can you translate into English the line:

   ```
   while continuation_condition: ?
   ```

4. Can you think of everyday activities in your life that are basically loops?
5. Can you think of examples of the accumulating pattern in everyday life? What are the equivalents for the "accumulators"?
6. What is an infinite loop?
7. Can a `while` loop be infinite? How?
8. What errors in the Python code could lead to an infinite loop?
9. What do the columns in the iteration table represent? What do the rows represent?