Lecture 9

Loops

A. Panchapakesan[1]

[1]Department of Computer Science
University of Ottawa

Introduction to Computing 1, Fall 2017

Outline

Outline

Outline

For-loops

What does this code print?

```
L = [1,2,3,4,5]
for i in L:
    print(i)
```

For-loops
Poll on Echo360

- The numbers 1 through 5
- The list `[1,2,3,4,5]`
- The list `[1,2,3,4,5]`, five times

Outline

List Slicing

What does this code print?

```
L = [1,2,3,4,5,6]
print(L[::-2])
```

List Slicing
Poll on Echo360

- `[6, 4, 2]`
- `[1, 3, 5]`
- `[5, 3, 1]`
- `[2, 4, 6]`

Outline

Memory Model

Consider this code

```
t = ([1,2,3], 5)
t[0][0] = 'a'
```

Memory Model

What happens when we run that code?

- `t` becomes `(['a', 2, 3], 5)`
- It blows up because tuples are immutable
- Something else

Outline

Outline

Consider This Code

It removes all 3s from a list

```
L = ['a', 2, 3, 3, 4, 5, 6, 2, 3, 5, 3]
for i in L:
    if i == 3:
        L.remove(i)
```

What is L now?

Poll on Echo360

- `['a', 2, 4, 5, 6, 2, 5]`
- Something else

A Closer Look at That Code

```
L = ['a', 2, 3, 3, 4, 5, 6, 2, 3, 5, 3]
for i in L:
    print(i)
    if i == 3:
        L.remove(i)
```

What's Happening There?

Let's look at the visualizer

- When you delete from a list, in-place
  - the iterator doesn't change
  - it knows the current index, and then goes to the next index (index+1)

What's Happening There?

Let's look at the visualizer

- When you delete from a list, in-place
  - the iterator doesn't change
  - it knows the current index, and then goes to the next index (index+1)

What's Happening There?

Let's look at the visualizer

- When you delete from a list, in-place
  - the iterator doesn't change
  - it knows the current index, and then goes to the next index (index+1)

Outline

Let's Change that Code Slightly

Don't actually run this code

Let's look at the visualizer

```
L = [1,2,3]
for i in L:
    L.append(i**2)
```

Outline

Outline

A For-loop Couldn't do this

```
L = ['a', 2, 3, 3, 4, 5, 6, 2, 3, 5, 3]
while 3 in L:
    L.remove(3)
```

That was the Easy Way

But we're ninjas, so let's try another way

```
L = ['a', 2, 3, 3, 4, 5, 6, 2, 3, 5, 3]
i = 0
while i<len(L):
    print("i is", i, "and L[i] is", L[i])
    if L[i] == 3:
        L.pop(i)
    else:
        i += 1
```

So What Did that For-loop Do?

```
L = ['a', 2, 3, 3, 4, 5, 6, 2, 3, 5, 3]
i = 0
while i<len(L):
    print("i is", i, "and L[i] is", L[i])
    if L[i] == 3:
        L.pop(i)
    i += 1
```

Outline

Adding to a List

Don't actually run this code

Let's look at the visualizer

```
L = [1,2,3]
i = 0
while i < len(L):
    L.append(L[i]**2)
```

Outline

Define `while`

- `while <condition>`
- Keep executing the body
  - until the condition is `False`
- If the condition is never `False`
  - execute the body infinitely (infinite loop)
- If the condition is never `True`
  - never execute the body
- Python does not have a `do-while` (but Java does)

Define `while`

- `while <condition>`
- Keep executing the body
  - until the condition is `False`
- If the condition is never `False`
  - execute the body infinitely (infinite loop)
- If the condition is never `True`
  - never execute the body
- Python does not have a `do-while` (but Java does)

Define `while`

- `while <condition>`
- Keep executing the body
  - until the condition is `False`
- If the condition is never `False`
  - execute the body infinitely (infinite loop)
- If the condition is never `True`
  - never execute the body
- Python does not have a `do-while` (but Java does)

Define `while`

- `while <condition>`
- Keep executing the body
  - until the condition is `False`
- If the condition is never `False`
  - execute the body infinitely (infinite loop)
- If the condition is never `True`
  - never execute the body
- Python does not have a `do-while` (but Java does)

Define `while`

- `while <condition>`
- Keep executing the body
  - until the condition is `False`
- If the condition is never `False`
  - execute the body infinitely (infinite loop)
- If the condition is never `True`
  - never execute the body
- Python does not have a `do-while` (but Java does)

Define `while`

- `while <condition>`
- Keep executing the body
  - until the condition is `False`
- If the condition is never `False`
  - execute the body infinitely (infinite loop)
- If the condition is never `True`
  - never execute the body
- Python does not have a `do-while` (but Java does)

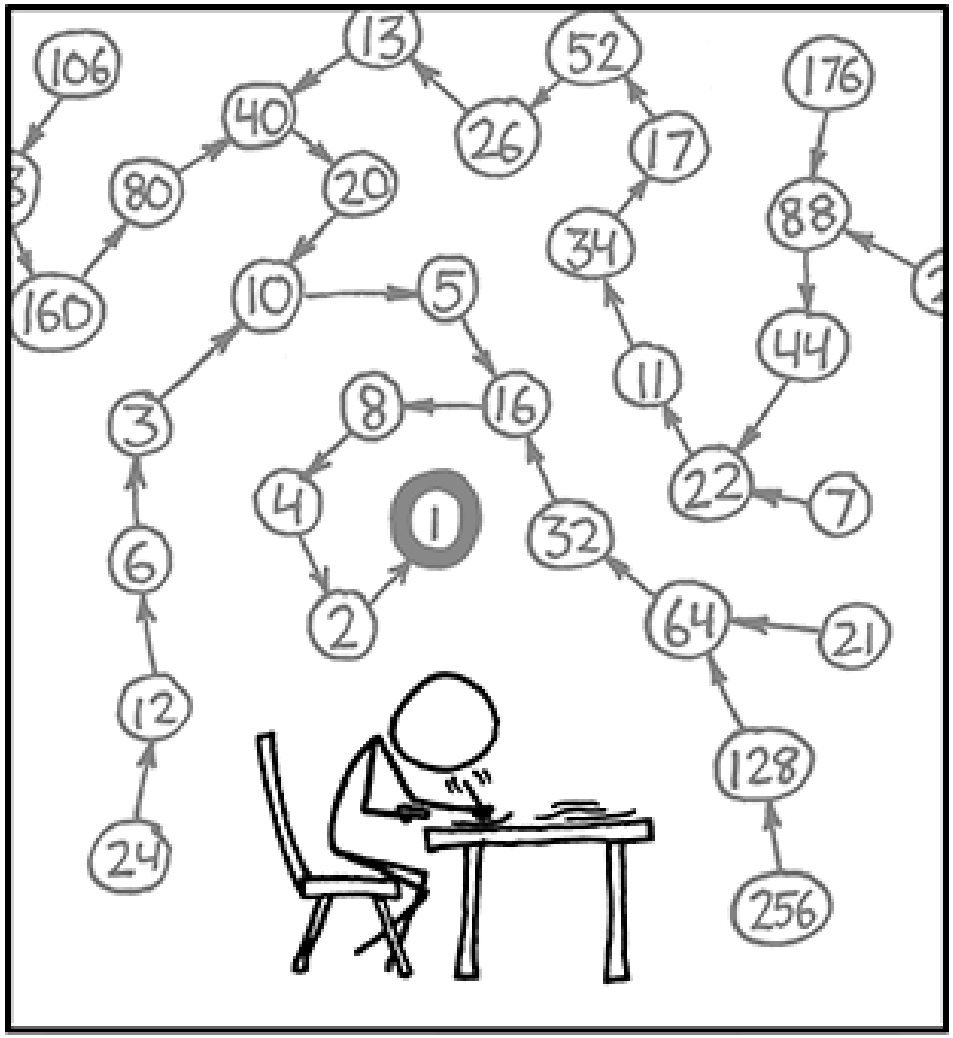
Define `while`

- `while <condition>`
- Keep executing the body
  - until the condition is `False`
- If the condition is never `False`
  - execute the body infinitely (infinite loop)
- If the condition is never `True`
  - never execute the body
- Python does not have a `do-while` (but Java does)

Define `while`

- `while <condition>`
- Keep executing the body
  - until the condition is `False`
- If the condition is never `False`
  - execute the body infinitely (infinite loop)
- If the condition is never `True`
  - never execute the body
- Python does not have a `do-while` (but Java does)

Collatz Conjecture



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG

Collatz Conjecture

- Pick a number
- If it's even
  - divide it by 2
- If it's odd
  - multiply by 3, and add 1
- Eventually, you'll reach 1
- No known proof
  - https://en.wikipedia.org/wiki/Collatz_conjecture

Collatz Conjecture

- Pick a number
- If it's even
  - divide it by 2
- If it's odd
  - multiply by 3, and add 1
- Eventually, you'll reach 1
- No known proof
  - https://en.wikipedia.org/wiki/Collatz_conjecture

Collatz Conjecture

- Pick a number
- If it's even
  - divide it by 2
- If it's odd
  - multiply by 3, and add 1
- Eventually, you'll reach 1
- No known proof
  - https://en.wikipedia.org/wiki/Collatz_conjecture

Collatz Conjecture

- Pick a number
- If it's even
  - divide it by 2
- If it's odd
  - multiply by 3, and add 1
- Eventually, you'll reach 1
- No known proof
  - https://en.wikipedia.org/wiki/Collatz_conjecture

Collatz Conjecture

- Pick a number
- If it's even
  - divide it by 2
- If it's odd
  - multiply by 3, and add 1
- Eventually, you'll reach 1
- No known proof
  - https://en.wikipedia.org/wiki/Collatz_conjecture

Collatz Conjecture

- Pick a number
- If it's even
  - divide it by 2
- If it's odd
  - multiply by 3, and add 1
- Eventually, you'll reach 1
- No known proof
  - https://en.wikipedia.org/wiki/Collatz_conjecture

Collatz Conjecture

- Pick a number
- If it's even
  - divide it by 2
- If it's odd
  - multiply by 3, and add 1
- Eventually, you'll reach 1
- No known proof
  - https://en.wikipedia.org/wiki/Collatz_conjecture

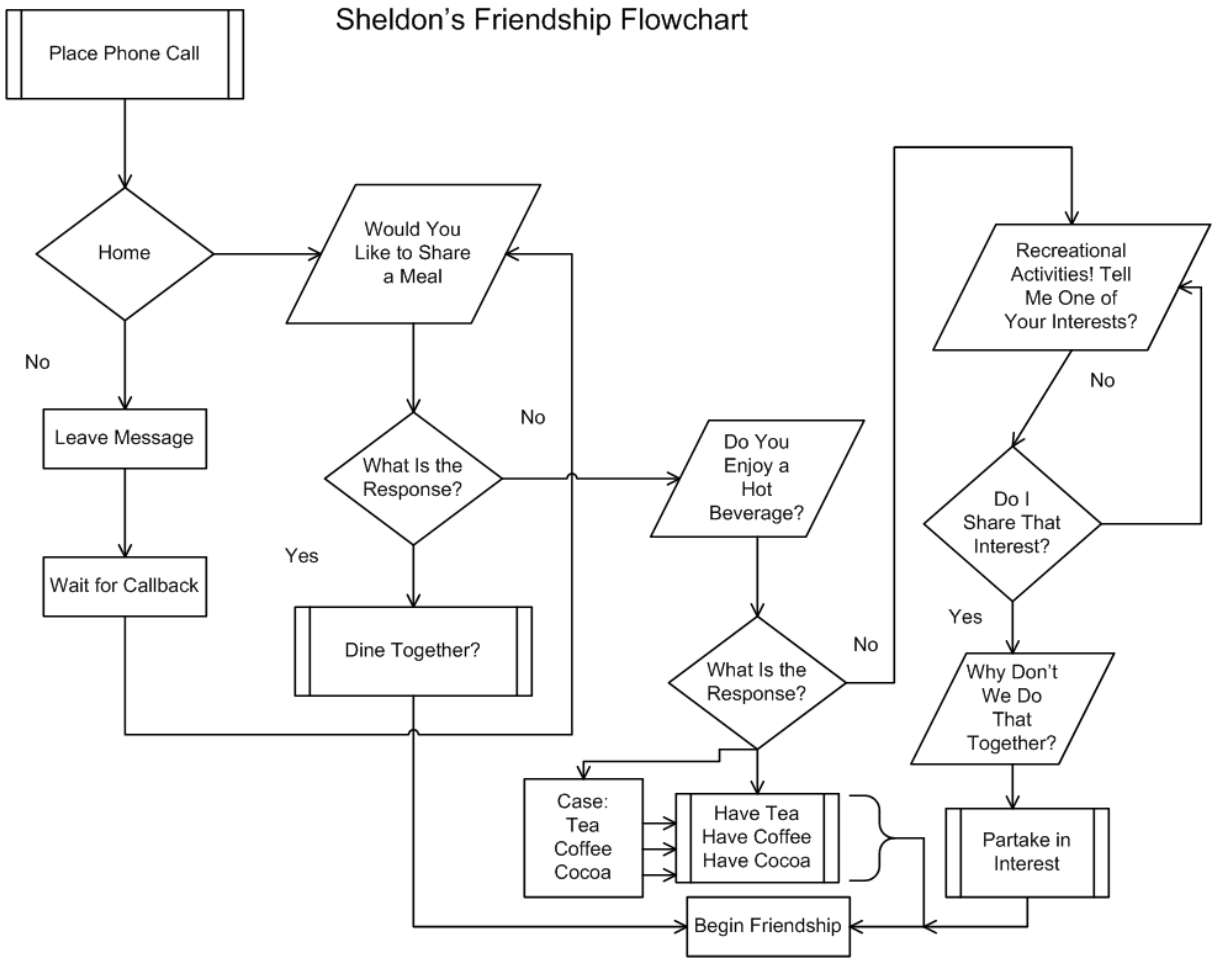
Collatz Conjecture

- Pick a number
- If it's even
  - divide it by 2
- If it's odd
  - multiply by 3, and add 1
- Eventually, you'll reach 1
- No known proof
  - https://en.wikipedia.org/wiki/Collatz_conjecture

Coding the Collatz Conjecture

```
n = 97
while n != 1:
    if not n%2:  # n is even
        n //= 2
    else:  # n is odd
        n = n*3 + 1
    print(n)
```

Infinite Loops



- https://www.youtube.com/watch?v=k0xgjUhEG3U