Indefinite Loops

Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

Recall: Two Types of `for` Loops

```
vals = [3, 15, 17, 7]
```

x

```
def sum(vals):
    result = 0
    for x in vals:
        result += x
    return result
```

element-based loop

```
vals[0] vals[1] vals[2] vals[3]
vals = [3,  15,  17,  7]
        0    1    2    3
```

i

```
def sum(vals):
    result = 0
    for i in range(len(vals)):
        result += vals[i]
    return result
```

index-based loop

Recall: Cumulative Computations

```
def sum(vals):
    result = 0          # the accumulator variable
    for x in vals:
        result += x     # gradually accumulates the sum
    return result

print(sum([10, 20, 30, 40, 50]))
```

See the video for a detailed trace of both versions!

x	result
	0
10	10
20	30
30	60
40	100
50	150

no more values in `vals`, so we're done
output: 150

Cumulative Computations with Strings

- Recall our recursive `remove_vowels` function:

```
def remove_vowels(s):
    if s == '':
        return ''
    else:
        removed_rest = remove_vowels(s[1:])
        if s[0] in 'aeiou':
            return removed_rest
        else:
            return s[0] + removed_rest
```

- Examples:

```
>>> remove_vowels('recurse')
'rcrs'
>>> remove_vowels('vowels')
'vwls'
```

Cumulative Computations with Strings (cont.)

- Here's one loop-based version:

```
def remove_vowels(s):
    result = ''             # the accumulator
    for c in s:
        if c not in 'aeiou':
            result += c     # accumulates the result
    return result
```

Cumulative Computations with Strings (cont.)

- Here's one loop-based version:

```
def remove_vowels(s):
    result = ''
    for c in s:
        if c not in 'aeiou':
            result += c
    return result
```

- Let's trace through `remove_vowels('vowels')`:

Cumulative Computations with Strings (cont.)

- Here's one loop-based version:

```
def remove_vowels(s):
    result = ''
    for c in s:
        if c not in 'aeiou':
            result += c
    return result
```

- Let's trace through `remove_vowels('vowels')`:

```
s = 'vowels'
```

Cumulative Computations with Strings (cont.)

- Here's one loop-based version:

```
def remove_vowels(s):
    result = ''
    for c in s:
        if c not in 'aeiou':
            result += c
    return result
```

- Let's trace through `remove_vowels('vowels')`:

```
s = 'vowels'
```

c	result
	''
'v'	'' + 'v' → 'v'
'o'	'v' (no change)
'w'	'v' + 'w' → 'vw'
'e'	'vw' (no change)
'l'	'vw' + 'l' → 'vwl'
's'	'vwl' + 's' → 'vwls'

Cumulative Computations with Strings (cont.)

- Here's one loop-based version:

```
def remove_vowels(s):
    result = ''
    for c in s:
        if c not in 'aeiou':
            result += c
    return result
```

- Let's trace through `remove_vowels('vowels')`:

```
s = 'vowels'
```

c	result
	''
'v'	'' + 'v' → 'v'
'o'	'v' (no change)
'w'	'v' + 'w' → 'vw'
'e'	'vw' (no change)
'l'	'vw' + 'l' → 'vwl'
's'	'vwl' + 's' → **'vwls'**

`for` Loops Are *Definite* Loops

- *Definite* loop = a loop in which the number of repetitions is *fixed* before the loop even begins.

- In a `for` loop, # of repetitions = `len(`*sequence*`)`

```
for variable in sequence:
    body of the loop
```

Indefinite Loops

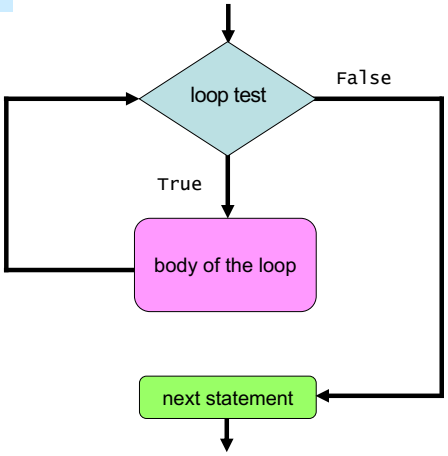
- Use an *indefinite loop* when the # of repetitions you need is:
  - not as obvious
  - impossible to determine before the loop begins
- In Python, we usually use a `while` loop for this.

`while` Loops

```
while loop test:
    body of the loop
```

Steps:

1. evaluate the loop test (a boolean expression)
2. if it's `True`, execute the statements in the body, and go back to step 1
3. if it's `False`, skip the statements in the body and go to the statement after the loop



Factorial Using a `while` Loop

- We don't need an indefinite loop, but we can still use `while`!

```
def fac(n):
    result = 1
    while n > 0:
        result *= n
        n = n - 1
    return result
```

- Let's trace `fac(4)`:

n	n > 0	result

Factorial Using a `while` Loop

- We don't need an indefinite loop, but we can still use `while`!

```
def fac(n):
    result = 1
    while n > 0:
        result *= n
        n = n - 1
    return result
```

- Let's trace `fac(4)`:

n	n > 0	result
4		

Factorial Using a `while` Loop

- We don't need an indefinite loop, but we can still use `while`!

```
def fac(n):
    result = 1
    while n > 0:
        result *= n
        n = n - 1
    return result
```

- Let's trace `fac(4)`:

n	n > 0	result
4		1
	4 > 0 (True)	1*4 = 4
3	3 > 0 (True)	4*3 = 12
2	2 > 0 (True)	12*2 = 24
1	1 > 0 (True)	24*1 = 24
0	0 > 0 (False)	

so we exit the loop and `return 24`

Factorial Four Ways!

recursion

```
def fac(n):
    if n == 0:
        return 1
    else:
        rest = fac(n-1)
        return n * rest
```

for loop

```
def fac(n):
    result = 1
    for x in range(1, n+1):
        result *= x
    return result
```

looping in assembly

```
00 read r1
01 setn r13 1
02 jeqz r1 6
03 mul r13 r13 r1
04 addn r1 -1
05 jumpn 02
06 write r13
07 halt
```

while loop

```
def fac(n):
    result = 1
    while n > 0:
        result *= n
        n = n - 1
    return result
```

Extreme Looping!

- What does this code do?

```
print('It keeps')
while True:                # always true!
    print('going and')
print('Phew! Done!')       # never gets here!
```

- An infinite loop!

output:

```
It keeps
going and
going and
going and
going and
going and
going and
going and
going and
```

Extreme Looping!

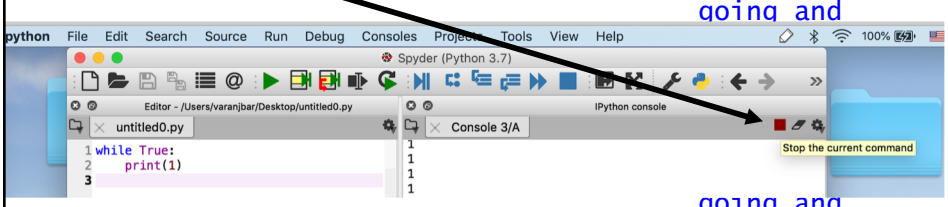
- What does this code do?

```
print('It keeps')
while True:                # always true!
    print('going and')
print('Phew! Done!')       # never gets here!
```

- An infinite loop!
- Use the **stop button** to stop it or **Control + C**.

output:

```
It keeps
going and
going and
```




```
going and
```

Choosing a Random Number

- Python's `random` module allows us to produce random numbers.
  - to use it, we need to import it:

```
import random
```

- `random.choice(vals)`
  - takes a sequence `vals`
  - randomly chooses one value from `vals` and returns it

- examples from the Shell:

```
>>> import random
>>> random.choice(range(7)) # random number from 0-6
5
>>> random.choice(range(7))
2
>>> random.choice(range(7))
4
```

Breaking Out of An Infinite Loop

```
import random

while True:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')

print('At last!')
```

Breaking Out of An Infinite Loop

```
import random

while True:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')

print('At last!')
```

A `break` statement causes a loop to end early.

- jumps to the line that comes after the loop

- Thus, the final two lines that are printed are:

```
Help!
At last!
```

- How could we count the number of repetitions?

Counting the Number of Repetitions

```
import random

count = 0
while True:
    count += 1
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')

print('At last! It took', count, 'tries to escape!')
```

User Input

- Getting a *string value* from the user:

  variable = `input(`*prompt*`)` where *prompt* is a string

- Getting an *integer value*:

  variable = `int(input(`*prompt*`))`

- Getting a *floating-point value*:

  variable = `float(input(`*prompt*`))`

- Getting an arbitrary non-string value (e.g., a list):

  variable = `eval(input(`*prompt*`))`

  - eval treats a string as an expression to be evaluated

- Examples:

```
name = input('What is your name? ')
count = int(input('possible points: '))
scores = eval(input('list of scores: '))
```

Using a `while True` Loop to Get User Input

```
import math

while True:
    val = int(input('Enter a positive number: '))
    if val > 0:
        break
    else:
        print(val, 'is not positive. Try again!')

result = math.sqrt(val)
print('result =', result)
```

Using a `while True` Loop to Get User Input

```
import math

while True:
    val = int(input('Enter a positive number: '))
    if val > 0:
        break
    else:
        print(val, 'is not positive. Try again!')

result = math.sqrt(val)
print('result =', result)
```

Using a `while True` Loop to Get User Input

```
import math

while True:
    val = int(input('Enter a positive number: '))
    if val > 0:
        break
    else:
        print(val, 'is not positive. Try again!')

result = math.sqrt(val)
print('result =', result)
```

How many values does this loop print?

```
a = 40
while a > 2:
    a = a // 2
    print(a - 1)
```

A. 2

B. 3

C. 4

D. 5

E. none of these

a > 2	a	prints

How many values does this loop print?

```
a = 40
while a > 2:
    a = a // 2
    print(a - 1)
```

A. 2

B. 3

C. 4

D. 5

E. none of these

a > 2	a	prints
	40	
True	20	19
True	10	9
True	5	4
True	2	1
False		

For what inputs does this function return `True`?

```
def mystery(n):
    while n != 1:
        if n % 2 != 0:
            return False
        n = n // 2
    return True
```

A. odd numbers

B. even numbers

C. multiples of 4

D. powers of 2

E. none of these

Try tracing these two cases:

mystery(12)	mystery(8)
n	n
12	8

For what inputs does this function return `True`?

```
def mystery(n):
    while n != 1:
        if n % 2 != 0:
            return False
        n = n // 2
    return True
```

A. odd numbers

B. even numbers

C. multiples of 4

D. **powers of 2**

E. none of these

Try tracing these two cases:

mystery(12)	mystery(8)
n	n
12	8
6	4
3	2
return False	1
	exit loop
	return True

What does this program output?

```
s = 'time to think! '
result = ''
for i in range(len(s)):
    if s[i - 1] == ' ':
        result += s[i]
print(result)
```

A. `tt`

B. `ttt`

C. `tothink!`

D. `timetothink!`

E. none of these

i	s[i-1]	s[i]	result

What does this program output?

```
s = 'time to think! '
result = ''
for i in range(len(s)):
    if s[i - 1] == ' ':
        result += s[i]
print(result)
```

A. tt

B. **ttt**

C. tothink!

D. timetothink!

E. none of these

i	s[i-1]	s[i]	result

What does this program output?

```
s = 'time to think! '
result = ''
for i in range(len(s)):
    if s[i - 1] == ' ':
        result += s[i]
print(result)
```

A. tt

B. **ttt**

C. tothink!

D. timetothink!

E. none of these

i	s[i-1]	s[i]	result
			''
0	' '	't'	**'t'**
1	't'	'i'	't'
2	'i'	'm'	't'
3	'm'	'e'	't'
4	'e'	' '	't'
5	' '	't'	**'tt'**
6	't'	'o'	'tt'
7	'o'	' '	'tt'
8	' '	't'	**'ttt'**
9	't'	'h'	'ttt'
10	'h'	'i'	'ttt'
11	'i'	'n'	'ttt'
12	'n'	'k'	'ttt'
13	'k'	'!'	'ttt'
14	'!'	' '	'ttt'

What does this program output?

```
s = 'time to think! '
result = ''
for i in range(len(s)):
    if s[i - 1] == ' ':
        result += s[i]
print(result)
```

Could you do the same thing using an *element*-based for loop?

```
s = 'time to think! '
result = ''
for c in s:
    if ________ == ' ':
        result += _____
print(result)
```

i	s[i-1]	s[i]	result
			''
0	' '	't'	't'
1	't'	'i'	't'
2	'i'	'm'	't'
3	'm'	'e'	't'
4	'e'	' '	't'
5	' '	't'	'tt'
6	't'	'o'	'tt'
7	'o'	' '	'tt'
8	' '	't'	'ttt'
9	't'	'h'	'ttt'
10	'h'	'i'	'ttt'
11	'i'	'n'	'ttt'
12	'n'	'k'	'ttt'
13	'k'	'!'	'ttt'
14	'!'	' '	'ttt'

What does this program output?

```
s = 'time to think! '
result = ''
for i in range(len(s)):
    if s[i - 1] == ' ':
        result += s[i]
print(result)
```

Could you do the same thing using an *element*-based for loop? ***no***

```
s = 'time to think! '
result = ''
for c in s:
    if ???? == ' ':
        result += c
print(result)
```

i	s[i-1]	s[i]	result
			''
0	' '	't'	't'
1	't'	'i'	't'
2	'i'	'm'	't'
3	'm'	'e'	't'
4	'e'	' '	't'
5	' '	't'	'tt'
6	't'	'o'	'tt'
7	'o'	' '	'tt'
8	' '	't'	'ttt'
9	't'	'h'	'ttt'
10	'h'	'i'	'ttt'
11	'i'	'n'	'ttt'
12	'n'	'k'	'ttt'
13	'k'	'!'	'ttt'
14	'!'	' '	'ttt'

Simpler

```
vals = [3, 15, 17, 7]
```

x

```
def sum(vals):
    result = 0
    for x in vals:
        result += x
    return result
```

element-based loop

More Flexible

```
vals = [3, 15, 17, 7]
```

vals[0] vals[1] vals[2] vals[3]

0 1 2 3

i

```
def sum(vals):
    result = 0
    for i in range(len(vals)):
        result += vals[i]
    return result
```

index-based loop