Loops & List

Intro2CS – week 3
Loops -- Motivation

• Sometimes we want to repeat a certain set of instructions more than once.

• The number of repetitions may be unknown in advance (depends on input).

• Two types of loops:
  – Repeat some instructions until some condition is met (while loops)
  – Repeat some instructions for every item in a sequence (for loops)
The while loop

while <boolean expression>:  
  <statements>

Start Here

Evaluate boolean expression

Evaluate boolean expression

False
Skip statements (resume rest of program)

True
Execute statements
A simple example

counter = 1

while counter <= 10:
    print(counter, end='   ')
    counter = counter + 1
A simple example

counter = 1

while counter <= 10:
    print(counter, end='  ')
    counter += 1
What if we forget?

```
counter = 1

while counter <= 10:
    print(counter, end='  ')
    counter += 1
```

The loop does not stop... This is a problem.
def is_prime(num):
    divisor = 2
    while divisor <= math.sqrt(num):
        if num % divisor == 0:
            return False
        divisor += 1
    return True
Break and Continue

Two statements to use in loops:

• break       exits the loop immediately.

• continue     skips remaining statements in loop
product = 1

print("Let's multiply some numbers")
while True:
    user_typed = input("Please enter a number " +
                        "(or type 'exit' to exit) ")
    if user_typed == "exit":
        break

num = float(user_typed)
# check if the user accidentally types 0
if num == 0:
    print("You probably did not mean that.")
    continue
product *= num

print("The product is:", product)
The for loop over sequences

def my_text = "a rat spat at a cat"

for letter in my_text:
    if letter == 'a':
        pass
    else:
        print(letter, end='')

A line that does not do anything.
How else can you write this?
Range objects represent a sequence of integers

range(stop)
  • represents ints from 0 to stop-1 (excluding stop)

range(start,stop)
  • represents ints from start to stop-1 (excluding stop)

range(start,stop,step)
  • represents ints from start to stop (excluding) with given step size
Counting

• So what do these print?

```python
for i in range(10):
    print(i, end=' ')

print()

for i in range(1, 11):
    print(i, end=' ')

print()

for i in range(10, 0, -1):
    print(i, end=' ')

print()
```
def is_prime_version2(num):
    for divisor in range(2, int(math.sqrt(num)) + 1):
        if num % divisor == 0:
            return False
    return True
Lists

• We may want to save a large amount of data. Example: the name of every person in class.

• Do we assign each to a different variable?

• What if we need to add a person?

• How do we process all this data in a loop?
Lists

Use [] to specify a list and assign it. Separate items with commas,

```
names = ["Alice", "Bob", "Charlie", "Dave", "Eve"]
```

Access items using their index:

```
print(names[0])  # Access to cell in list. Indices start at 0
print(names[-1]) # Negative indices: count from the end.
print(len(names)) # Length of the list. Last index is length-1.
names[1] = "Robert" # We can assign to cells (like regular variables)
print(names)
```

Output:

```
Alice
Eve
5
['Alice', 'Robert', 'Charlie', 'Dave', 'Eve']
```
Lists

An empty list:

```python	names = []
```

Other ways to create lists through the list “constructor”:

```python
list()                      []
list("word")               ['w', 'o', 'r', 'd']
list(range(5))             [0, 1, 2, 3, 4]
```
Multiplication and addition

\[
\text{print}([1, 2, 3] * 3)
\]
\[
\text{print}([1, 2, 3] + [4, 5, 6, 7])
\]

membership, deleting, appending

\[
\text{names} = ['Alice', 'Bob', 'Charlie', 'Dave']
\]
\[
\text{print}('Bob' in names)
\]
\[
\text{del} names[1:3]
\]
\[
\text{print(names)}
\]
\[
\text{names}.append('Eve')
\]
\[
\text{print(names)}
\]
Slicing

numbers = list(range(10))

numbers[:]  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

numbers[3:]  
[3, 4, 5, 6, 7, 8, 9]

numbers[:3]  
[0, 1, 2]

numbers[1:3]  
[1, 2]

numbers[::2]  
[0, 2, 4, 6, 8]

numbers[:::-1]  
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Assign to slices:
numbers[1:3] = ['a', 'b', 'c', 'd']

[0, 'a', 'b', 'c', 'd', 3, 4, 5, 6, 7, 8, 9]
loops over lists

• We can loop over lists just like any other sequence:

```python
for element in my_list:
    print(element)
```

• Or:

```python
for index in range(len(my_list)):
    print(my_list[index])
```

(less elegant, but works)
Finding all primes

```python
def all_primes_up_to(m):  # excluding m
    primes = list()
    for i in range(2, m):
        if is_prime(i):
            primes.append(i)
    return primes
```

```bash
>>> print(all_primes_up_to(100))
```
The Sieve of Eratosthenes

- Mark suspected prime numbers in a list

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21|
| T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T |

- The idea: once we find a prime number, we can erase all its multiples as primes.

- 2 is prime.

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21|
| T | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T |

- Next “suspect” is also a prime (why?)
The Sieve of Eratosthenes

Not Prime, so multiples already erased!

In fact, we are done. Why?

Hint: $5 > \sqrt{21}$
The Sieve of Eratosthenes

def all_primes_sieve(m):  # excluding m
    primes = list()

    marked_prime = [True]*m
    marked_prime[0] = False
    marked_prime[1] = False

    for candidate in range(2, m):
        if marked_prime[candidate]:
            primes.append(candidate)
            for i in range(candidate*2, m, candidate):
                marked_prime[i] = False

    return primes
```python
def pyramid(height):
    for row in range(height):
        for _ in range(height-row-1):
            print(" ", end="")
        for _ in range(row*2+1):
            print("*", end="")
    print()

pyramid(10)
```

```
*  
*** 
***** 
******* 
********* 
********** 
*********** 
************ 
************* 
************** 
*************** 
*************** 
************** 
************* 
************ 
*********** 
********** 
********* 
***** 
*** 
*  
```
Nested lists

• We can place lists inside each other.

```python
records = [['Joe', 60], ['Jane', 95]]
print(records)
records.append(['Danny', 97])
records.append(['Tammy', 100])
print(records)

print(records[2])
print(records[2][0])
print(records[2][1])
```

```
[['Joe', 60], ['Jane', 95]]
[['Joe', 60], ['Jane', 95], ['Danny', 97], ['Tammy', 100]]
['Danny', 97]
Danny
97
```