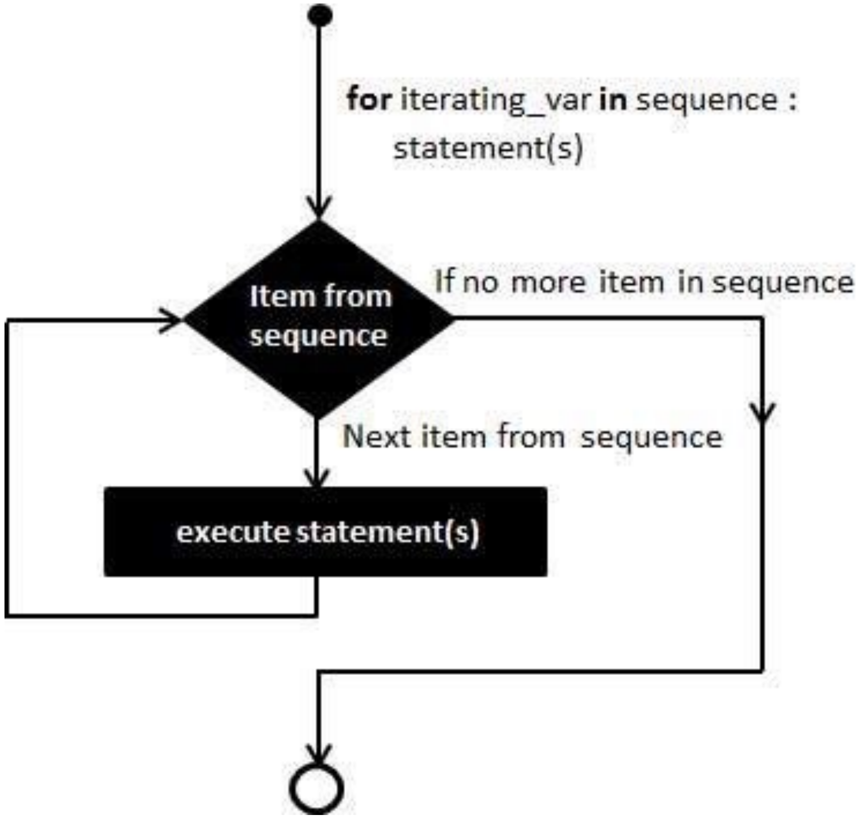
SAMS
Programming - Section C

Lecture 3:
Intro to loops



July 7, 2017

Approximate value of floats

Math module

My first ever program

```
************
***********
**********
*********
********
*******
******
*****
****
***
**
*
```

My first ever program

```
print(“************”)
print(“***********”)
print(“**********”)
print(“*********”)
print(“********”)
print(“*******”)
print(“******”)
print(“*****”)
print(“****”)
print(“***”)
print(“**”)
print(“*”)
```

There is a
better way!

Loops give you wings.

2 types of loops in Python

for loop

while loop

for loop

```
for var-name in sequence:
    loop-body
```

```
repeat 5 times:
    print("Hello")
```

≡

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

↓

(but this is not valid Python syntax)

loop body

(can be as many lines as you want)

iteration: a single execution of the instructions in the loop body.

for loop

```
for var-name in sequence:
    loop-body
```

```
for i in [1, 2, 3, 4, 5]:
    print(“Hello”)
```

→ list (a data type in Python)

Same as:

```
    print(“Hello”)
    print(“Hello”)
    print(“Hello”)
    print(“Hello”)
    print(“Hello”)
```

1st iteration:	i = 1
2nd iteration:	i = 2
3rd iteration:	i = 3
4th iteration:	i = 4
5th iteration:	i = 5

for loop

```
for var-name in sequence:
    loop-body
```

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

Same as:

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

1st iteration:	i = 1
2nd iteration:	i = 2
3rd iteration:	i = 3
4th iteration:	i = 4
5th iteration:	i = 5

for loop

```
for var-name in sequence:
    loop-body
```

range(n) ≈ [0, 1, 2, …, n-1]

```
for i in [0, 1, 2, 3, 4]:
    print(i)
```

≡

```
for i in range(5):
    print(i)
```

for loop

for *var-name* **in** *sequence*:
    loop-body

```
def sumToN(n):
    total = 0
    for i in range(n+1):
        total += i
    return total


print(sumToN(4))
```

```
total = 0
total += 0
total += 1
total += 2
total += 3
total += 4
return total
```

for loop

```
for var-name in sequence:
    loop-body
```

range(m, n) ≈ [m, m+1, m+2, …, n-1]

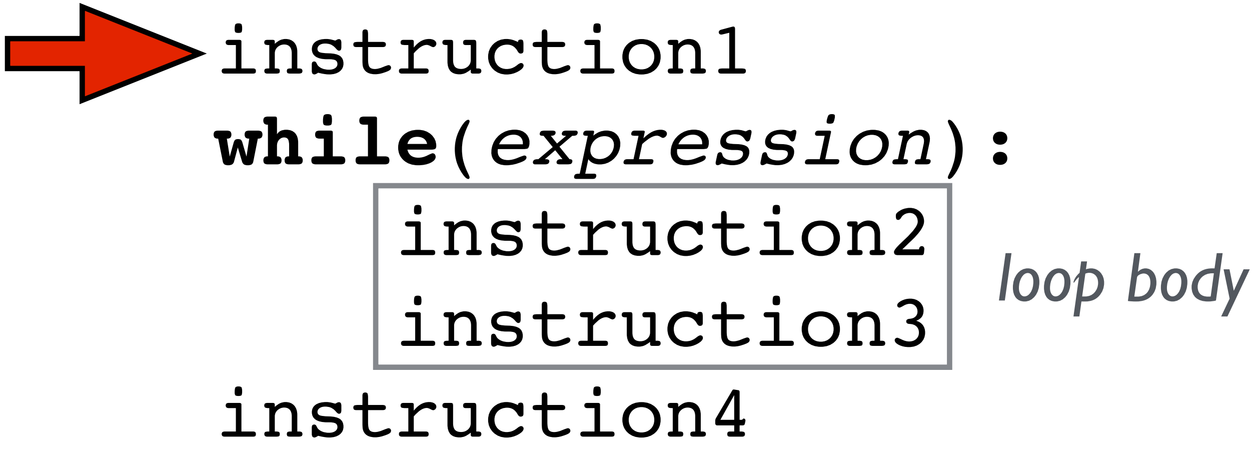
```
def sumFromMToN(m, n):
    total = 0
    for i in range(m, n+1):
        total += i
    return total
```

2 types of loops in Python

for loop

➡ **while** loop

while loop



```
instruction1
while(expression):
    instruction2
    instruction3
instruction4
```

loop body

The code in the loop body should change something related to the *`expression`*.

while loop example

```
def getPositiveInteger():
    userInput = 0
    while (userInput <= 0):
        userInput = int(input("Enter a positive integer: "))
    return userInput
```

while loop

Repeating a block a certain number of times:

```
repeat 5 times:
   instruction1
   instruction2
```

(but this is not valid Python syntax)

≡

```
counter = 1

while(counter <= 5):
   instruction1
   instruction2
   counter += 1
```

Never use while loops to do this. Use for loops.

while loop example

```
def countToN(n):
    counter = 1
    while (counter <= n):
        print(counter)
        counter += 1
```

1st iteration:	counter = 1
2nd iteration:	counter = 2
3rd iteration:	counter = 3
4th iteration:	counter = 4
⋮	
n’th iteration:	counter = n

while loop example

```
def sumToN(n):
    counter = 1
    total = 0
    while (counter <= n):
        total += counter
        counter += 1
    return total
```

while loop example

```
def sumFromMToN(m, n):
    counter = m
    total = 0
    while (counter <= n):
        total += counter
        counter += 1
    return total
```

Again: never use while loops to do these.
Use for loops.

Common Loop Bug 1

Off by 1 error

```
def sumToN(n):
    total = 0
    counter = 0
    while (counter <= n):
        counter += 1
        total += counter
    return total
```

Loop conditions that results in the loop body being executed either:

- 1 time too few
- 1 time too many

Manually check first and last iterations!

Common Loop Bug 2

Infinite Loops

```
counter = 1
while (counter < 10):
    # Do some awesome complicated computation
    # ...
    # Then forget to increment counter
```

In the body, you have to change something related to the *condition* being checked.

for loop vs **while** loop

```
for i in range(10):
    # some code
```

```
i = 0
while (i < 10):
    # some code
    i += 1
```

For loop is the right choice here!

Use **while** loop when the number of iterations is ***indefinite***.

e.g. continue to do something <u>until</u> a certain event

Example: leftmost digit

Write a function that

- takes an integer n as input,
- returns its leftmost digit.

e.g. 409283402013 should return 4

Idea:

Repeatedly get rid of rightmost digit until one digit is left.

```
def leftmostDigit(n):
    while (n >= 10):
        n = n // 10
    return n
```

Example: leftmost digit

Write a function that

- takes an integer n as input,
- returns its leftmost digit.

e.g. 409283402013 should return 4

Idea:

Repeatedly get rid of rightmost digit until one digit is left.

```
def leftmostDigit(n):
    n = abs(n)
    while (n >= 10):
        n //= 10
    return n
```

Exercise: Testing primality

Write a function that:

- Gets an integer input
- Returns True if the integer is prime
- Returns False otherwise

prime:

- greater than 1,
- is only divisible by 1 and itself

Exercise: Testing primality

Steps to follow

- Find a mental picture of the solution

- Write an algorithm

- Write the code

- TEST!

- Fix the bugs (if any)

Exercise: Testing primality

- Find a mental picture of the solution

Example input: 961748941

How would **you** figure out the answer
if you had *paper*, *pencil*, and *calculator*?

Exercise: Testing primality

Steps to follow

- Find a mental picture of the solution

- Write an algorithm

- Write the code

- TEST!

- Fix the bugs (if any)

Exercise: Testing primality

- Write an algorithm

Algorithm:

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

Exercise: Testing primality

- Write an algorithm

Algorithm:

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

Exercise: Testing primality

Steps to follow

- Find a mental picture of the solution

- Write an algorithm

- Write the code

- TEST!

- Fix the bugs (if any)

Exercise: Testing primality

- Write the code

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
```

Exercise: Testing primality

- Write the code

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
    for possibleFactor in range(2, n):
```

Exercise: Testing primality

- Write the code

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
    for possibleFactor in range(2, n):
        # Check if possibleFactor divides n
```

Exercise: Testing primality

- Write the code

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
    for possibleFactor in range(2, n):
        if (n % possibleFactor == 0): return False
```

Exercise: Testing primality

- Write the code

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
    for possibleFactor in range(2, n):
        if (n % possibleFactor == 0): return False
    return True
```

Exercise: Testing primality

- Write the code

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
    if (n < 2): return False
    for possibleFactor in range(2, n):
        if (n % possibleFactor == 0): return False
    return True
```

Exercise: Testing primality

Steps to follow

- Find a mental picture of the solution

- Write an algorithm

- Write the code

- TEST!

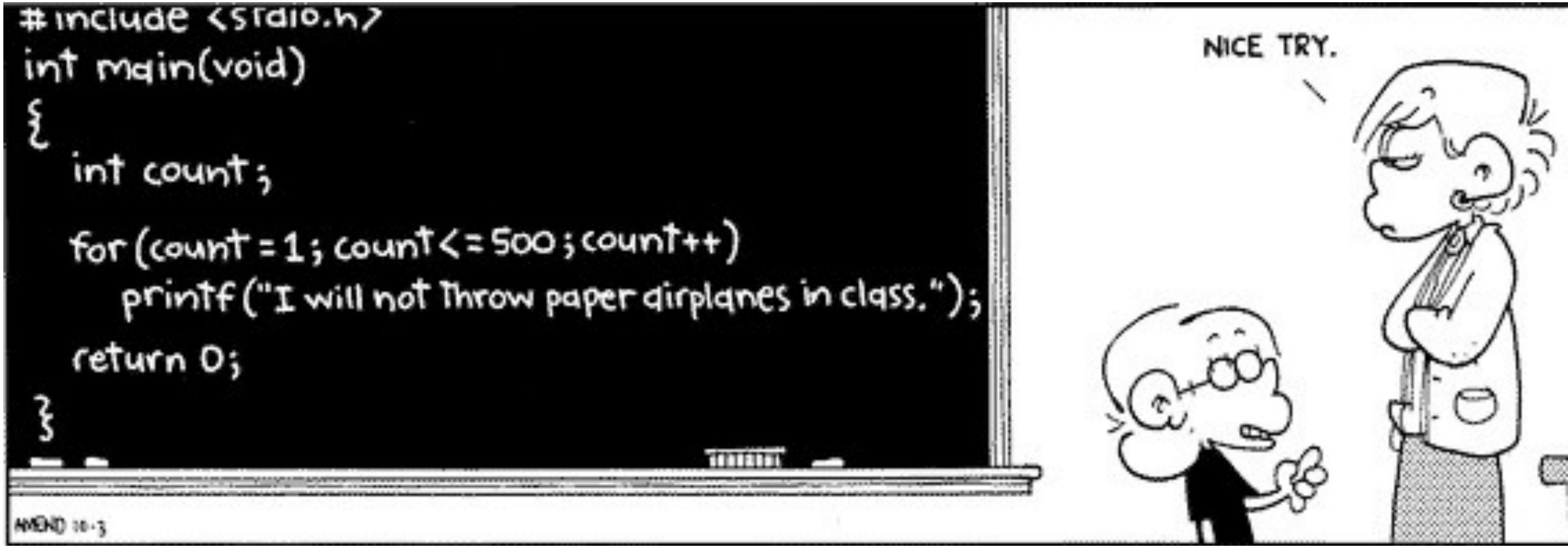
- Fix the bugs (if any)

Exercise: Testing primality

- TEST!

```
def testIsPrime():
    assert(not isPrime(0))
    assert(not isPrime(1))
    assert(not isPrime(-1))
    assert(isPrime(2))
    assert(not isPrime(-2))
    assert(isPrime(3))
    assert(not isPrime(4))
    assert(isPrime(5))
    assert(not isPrime(6))
    assert(not isPrime(-3))
    assert(isPrime(251))
    assert(not isPrime(15251))
    print("Passed all tests!")
```

Passes all tests!



```
#include <stdio.h>
int main(void)
{
   int count;

   for (count = 1; count <= 500; count++)
      printf ("I will not Throw paper airplanes in class.");

   return 0;
}
```