15-112
Fundamentals of Programming
Week 1 - Lecture 3:
Loops
Basic Building Blocks

**Statements**
Tell the computer to do something.

**Data Types**
Data is divided into different types.

**Variables**
Allows you to store data and access stored data.

**Operators**
Allows you to manipulate data.

**Conditional Statements**
Executes statements if a condition is satisfied.

**Functions**
Mini self-contained programs.

**Loops**
Execute a block of code multiple times.
Loops give you wings!
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!
2 types of loops in Python

- **while loop**
- **for loop**
while loop

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

The code in the block should change something related to the expression.

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

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

while (x < 5):
    print("Value of x is", x)
    x += 10
    print("This line will be printed!")

print("bye")
while loop

Repeating a block a certain number of times:

counter = 1

while (counter <= 10):
    instruction1
    instruction2
    counter += 1

But never use while loops to do this. Use for loops.
def countToN(n):
    counter = 1
    while(counter <= n):
        print(counter)
        counter += 1
def sumToN(n):
    counter = 1
    total = 0
    while (counter <= n):
        total += counter
        counter += 1
    return total
def sumFromMToN(m, n):
    counter = m
    total = 0
    while (counter <= n):
        total += counter
        counter += 1
    return total

Again: never use while loops to do this. 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 the first and last iterations!
In the body you have to change something related to the *condition* being checked.
Write a function that
- takes an integer \( n \) as input,
- returns its leftmost digit.

\[
\text{e.g. } 409283402013 \quad \text{should return} \quad 4
\]

**Idea:**
Repeatedly get rid of rightmost digit until one digit is left.

```python
def leftmostDigit(n):
    while (n >= 10):
        n = n // 10
    return n
```
Write a function that
- takes an integer \( n \) as input,
- returns its leftmost digit.

\[
\text{e.g. } \quad 409283402013 \quad \text{should return} \quad 4
\]

**Idea:**
Repeatedly get rid of rightmost digit until one digit is left.

```python
def leftmostDigit(n):
    n = abs(n)
    while (n >= 10):
        n = n // 10
    return n
```
A number $m \geq 0$ is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number $n$ as input,
- returns the $n$’th Awesome number. (counting starts from 0)
Example: n’th Awesome number

A number \( m \geq 0 \) is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number \( n \) as input,
- returns the \( n \)’th Awesome number. (counting starts from 0)

Pictorial solution:

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots \\
\uparrow & & & & & & & & & & \\
is Awesome? & & & & & & & & & & \\
\text{yes} & & & & & & & & & &
\end{array}
\]

\( n = 4 \)

\( \text{found} = 1 \)
Example: n’th Awesome number

A number \( m \geq 0 \) is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number \( n \) as input,
- returns the \( n \)’th Awesome number. (counting starts from 0)

Pictorial solution:

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots \\
\uparrow \\
\end{array}
\]

is Awesome? \[ \text{no} \]
A number \( m \geq 0 \) is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that  
- takes a positive number \( n \) as input,  
- returns the \( n \)’th Awesome number. (counting starts from 0)

**Pictorial solution:**

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots \\
\uparrow & & & & & & & & & & \\
\text{is Awesome?} & & & & & & & & \text{no} & & \\
\end{array}
\]
A number \( m \geq 0 \) is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number \( n \) as input,
- returns the \( n \)’th Awesome number.  

**Pictorial solution:**

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots \\
\uparrow & & & & & & & & & & \\
\text{is Awesome?} & & & & & & & & & & \\
\text{yes} & & & & & & & & & & \\
\end{array}
\]

\( n = 4 \)

found = 2
A number \( m \geq 0 \) is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number \( n \) as input,
- returns the \( n \)’th Awesome number. (counting starts from 0)

**Pictorial solution:**

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots
\end{array}
\]

\[
\uparrow
\]

is Awesome?

no

\( n = 4 \)

found = 2
A number $m \geq 0$ is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number $n$ as input,
- returns the $n$’th Awesome number. (counting starts from 0)

**Pictorial solution:**

0 1 2 3 4 5 6 7 8 9 ...  

is Awesome?  
yes  

$n = 4$  
found = 3
Example: n’th Awesome number

A number \( m \geq 0 \) is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number \( n \) as input,
- returns the \( n \)’th Awesome number. (counting starts from 0)

Pictorial solution:

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots \\
\end{array}
\]

\[
\uparrow
\]

is Awesome?

\[
\text{yes}
\]

\[
\begin{array}{cccccccccc}
n = 4 \\
\text{found} = 4
\end{array}
\]
A number \( m \geq 0 \) is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number \( n \) as input,
- returns the \( n \)’th Awesome number. (counting starts from 0)

Pictorial solution:

\[
0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad \ldots
\]

\[
\uparrow
\]

is Awesome?

\[
\text{no}
\]

\[
n = 4
\]

\[
\text{found} = 4
\]
A number \( m \geq 0 \) is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number \( n \) as input,
- returns the \( n \)’th Awesome number. (counting starts from 0)

Pictorial solution:

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots \\
\end{array}
\]

\( n = 4 \)

is Awesome?

\( \text{found} = 4 \)

\( \text{no} \)
A number $m \geq 0$ is called “Awesome” if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number $n$ as input,
- returns the $n$’th Awesome number. (counting starts from 0)

Pictorial solution:

```
0 1 2 3 4 5 6 7 8 9 ...  

is Awesome?   n = 4
found = 5
yes  →  return 9
```
Example: n’th Awesome number

Pictorial solution:

0 1 2 3 4 5 6 7 8 9 ... 

is Awesome? found = 5

yes → return 9 

Algorithm:

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number
Example: n’th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```python
def nthAwesome(n):
```
Example: n’th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```python
def nthAwesome(n):
    found = 0
```
Example: n’th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ...:
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```python
def nthAwesome(n):
    found = 0
    guess = 0
```
Example: n’th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```python
def nthAwesome(n):
    found = 0
    guess = -1
```
Example: n’th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ...:
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```python
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
```
Example: n’th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ...:
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```python
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
        guess += 1
```
Example: n’th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```python
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):  
        guess += 1
        if (isAwesome(guess)): 
```
Example: n’th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```python
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
        guess += 1
        if (isAwesome(guess)):
            found += 1
```
Example: n’th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```python
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
        guess += 1
        if (isAwesome(guess)):
            found += 1
    return guess
```
Example: n’th Awesome number

```python
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
        guess += 1
        if isAwesome(guess):
            found += 1
    return guess

def isAwesome(m):
    return ((m % 3) == 0) or ((m % 5) == 0)
```

```python
def nthAwesome(n):
    found = 0
    guess = 0
    while (found <= n):
        if (isAwesome(guess)):
            found += 1
        guess += 1
    return guess - 1
```
2 types of loops in Python

while loop

for loop
for loop

for var-name in sequence:
  loop-body

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

list (a data type in Python)

1st iteration:  x = 1
2nd iteration:  x = 2
3rd iteration:  x = 3
4th iteration:  x = 4
5th iteration:  x = 5
for loop

for var-name in sequence:
    loop-body

for x in "Hello":
    print(x)

A string is a sequence too

1st iteration: x = "H"
2nd iteration: x = "e"
3rd iteration: x = "l"
4th iteration: x = "l"
5th iteration: x = "o"
for loop

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

range(n) ≈ [0, 1, 2, ..., n-1]

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

```
for x in range(5):
    print(x)
```
for var-name in sequence:
    loop-body

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

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

For loop is the right choice here!
**for loop**

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

range(m, n) ≈ \([m, m+1, m+2, \ldots, n-1]\)

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

for var-name in sequence:
    loop-body

range(m, n, k) ≈ [m, m+k, m+2k, …]

def sumEveryKthFromMToN(m, n, k):
    total = 0
    for x in range(m, n+1, k):
        total += x
    return total
def sumOfOddsFromMToN(m, n):
    total = 0
    for x in range(m, n+1):
        if (x % 2 == 1):
            total += x
    return total
```python
def sumOfOddsFromMToN(m, n):
    total = 0
    for x in range(m, n+1):
        if (x % 2 == 1):
            total += x
    return total

def sumOfOddsFromMToN(m, n):
    if (m % 2 == 0): m += 1
    total = 0
    for x in range(m, n+1, 2):
        total += x
    return total
```
def sumOfOddsFromMToN(m, n):
    if (n % 2 == 0): n -= 1
    total = 0
    for x in range(n, m-1, -2):
        total += x
    return total

Unclear code!!!
**for loop vs while loop**

```
for x in range(10):
    print(x)
```

```
x = 0
while (x < 10):
    print(x)
    x += 1
```

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

- e.g. continue to do something **until** a certain event
Exercise: Testing primality

Write a function that:
- Gets a positive 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

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

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

- 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.

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

- 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.

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

- 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.

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

```python
def isPrime(n):
    if (n < 2): return False
    for possibleFactor in range(2, n):
        if (n % possibleFactor == 0): return False
    return True
```
Start thinking about running time

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

How many iterations?

In the worst case?
(worst possible \( n \))

\(~ n\)

What if the input is

2037035976334486086268445688409378161051468393665936250636140449354381299763336706183397371

(length of the input = number of digits = 90 \(\sim\) log \(n\))
Start thinking about running time

```python
def fasterIsPrime(n):
    if (n < 2): return False
    maxFactor = round(n**0.5)
    for x in range(2, maxFactor + 1):
        if(n % x == 0): return False
    return True
```

How many iterations?

In the worst case?
(worst possible n)

~ n**0.5
Example: Find the n’th prime

Write a program that:
- Gets a positive integer \( n \) as input
- Returns the \( n \)’th prime number

- Let found = 0
- Go through every number 2, 3, 4, 5, ... :
  - if the number is prime, increment found
- When found > \( n \), return the corresponding prime

Remember: We start counting from 0.
Example: Find the n’th prime

- Let found = 0
- Go through every number 2, 3, 4, 5, ... :
  - if the number is prime, increment found
- When found > n, return the corresponding prime

```python
def nthPrime(n):
    found = 0
    guess = 0
    while (found <= n):
        guess += 1
        if (isPrime(guess)):
            found += 1
    return guess
```

Need to use while loop
Example: The factoring problem

Write a function that:
- gets a positive integer as input
- returns the smallest \textit{factor} \neq 1

\textit{factor}: divides the integer with no remainder.

Exercise
Why you should care about this problem:

If there is an efficient program to solve the \textit{factoring problem} can break most cryptographic systems used on the internet!
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
#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;
}