NUS | Computing

National University of Singapore

# IT5001 Software Development Fundamentals

3. Control Structures

Sirigina Rajendra Prasad

(Slide Credit: Prof. Alan, SoC, NUS)

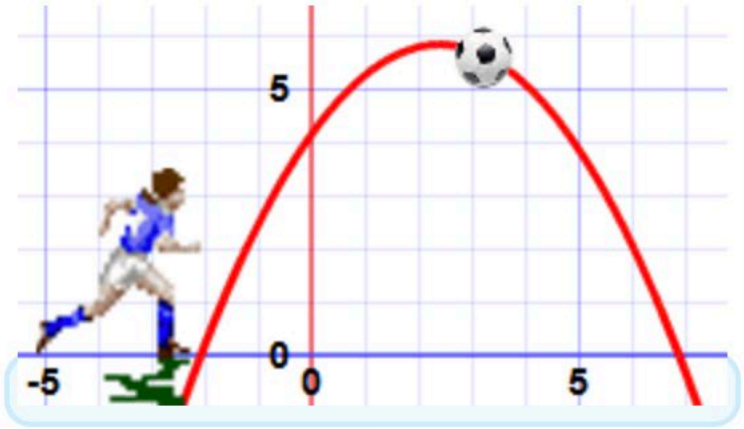
# Example: Solving a Quadratic Equation

An example of a **Quadratic Equation**:

*this makes it Quadratic*

$$5x^2 + 3x + 3 = 0$$

Quadratic Equations make nice curves, like this one:



# Example: Solving a Quadratic Equation

## Standard Form

The **Standard Form** of a Quadratic Equation looks like this:

$$ax^2 + bx + c = 0$$

- **a**, **b** and **c** are known values. **a** can't be 0.
- "**x**" is the **variable** or unknown (we don't know it yet).

# Example: Solving a Quadratic Equation

- Remember what we learned in high school…

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Let's try to implement it in Python

```
from math import sqrt

def solve_qe(a,b,c):
    delta = b**2 - 4*a*c
    ans1 = (-b + sqrt(delta))/(2*a)
    ans2 = (-b - sqrt(delta))/(2*a)
    print("The two solutions are " + str(ans1)
          + " and " + str(ans2))
```

```
>>> solve_qe(1,5,6)
The two solutions are -2.0 and -3.0
>>> solve_qe(1,4,4)
The two solutions are -2.0 and -2.0
>>>
```

# However…

```
>>> solve_qe(1,-5,6)
The two solutions are 3.0 and 2.0
>>> solve_qe(1,1,8)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    solve_qe(1,1,8)
  File "C:\Users\dcschl\Google Drive\Courses\YSC22
21\Lectures\solve_qe1.py", line 5, in solve_qe
    ans1 = (-b + sqrt(delta))/(2*a)
ValueError: math domain error
```

- Why?

```
from math import sqrt

def solve_qe(a,b,c):
    delta = b**2 - 4*a*c
    ans1 = (-b + sqrt(delta))/(2*a)
    ans2 = (-b - sqrt(delta))/(2*a)
    print("The two solutions are " + str(ans1)
          + " and " + str(ans2))
```

delta = 25-24 = 1 > 0

```
>>> solve_qe(1,-5,6)
The two solutions are 3.0 and 2.0
>>> solve_qe(1,1,8)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    solve_qe(1,1,8)
  File "C:\Users\dcschl\Google Drive\Courses\YSC22
21\Lectures\solve_qe1.py", line 5, in solve_qe
```

delta = 1 – 32 = -31 < 0

# Example: Solving a Quadratic Equation

- Remember what we learned in high school…

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- If delta < 0
  - The equation has no real solution

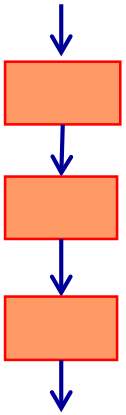
- So we cannot call sqrt() if delta is negative

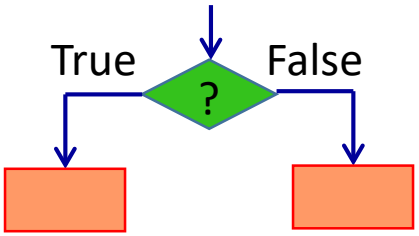
# Control Structures

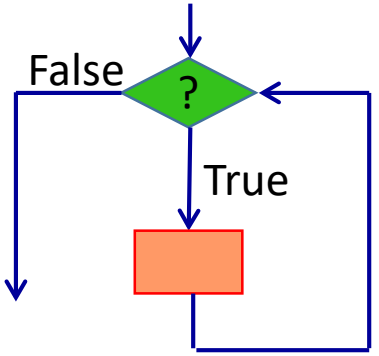
The basic building blocks of programming

# Control Structures

| | |
|---|---|
| Sequence | • Default |
| Selection | • Also called branching |
| Repetition | • Also called loop |

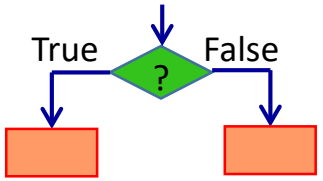




# Making Choices

# Control Structure: Selection



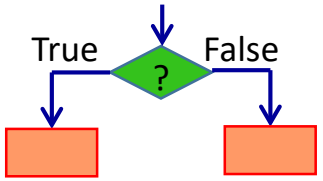

```
If (a condition is true)
  Do A
Else
  Do B
```

Can be MORE THAN one single instruction

- For example:

```
If (I have $10000000000000000)
  Buy a car
  Eat a lot of buffets
  Go travel
  Quit NUS!
Else
  Be good and study
```

# Control Structure: Selection




```
If (a condition is true)
  Do A
Else
  Do B
```

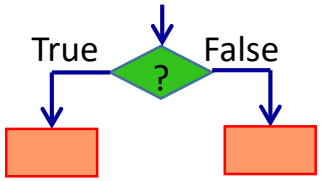
Can be MORE THAN one single instruction

- For example:

```
If (I have $10000000000000000)
  If (I am heartless)
    Buy a car
    Eat a lot of buffets
    Go travel
    Quit NUS!
  Else
       donate all the money to charity
Else
  Be good and study
```

Nested "if"

# Control Structure: Selection



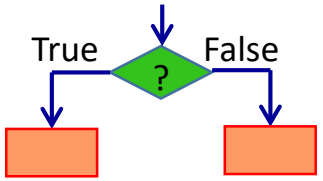

```
If (a condition is true)
  Do A
Else
  Do B
```

Can be **WITHOUT** "else"

- For example:

```
If (I have $10000000000000000)
  Buy a car
  Eat a lot of buffets
  Go travel
  Quit NUS!
Else
  Be good and study
```

# Control Structure: Selection




```
If (a condition is true)
  Do A
Else
  Do B
```

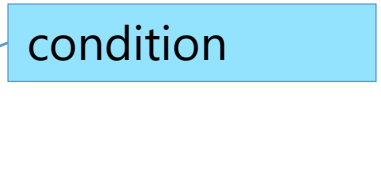
Can be MORE THAN one single instruction

- For example:

```
If (I have $10000000000000000)
  Buy a car
  Eat a lot of buffets
  Go travel
  Quit NUS!
Else
  Be good and study
```

# Condition

```
def solve_qe(a,b,c):
    delta = b**2 - 4*a*c
    if delta >= 0:
        ans1 = (-b + sqrt(delta))/(2*a)
        ans2 = (-b - sqrt(delta))/(2*a)
        print("The two solutions are " + str(ans1)
              + " and " + str(ans2))
    else:
        print("The equation has no real root")
```



# If the condition is True

```
def solve_qe(a,b,c):
    delta = b**2 - 4*a*c
    if delta >= 0:
        ans1 = (-b + sqrt(delta))/(2*a)
        ans2 = (-b - sqrt(delta))/(2*a)
        print("The two solutions are " + str(ans1)
              + " and " + str(ans2))
    else:
        print("The equation has no real root")
```

# If the condition is False

```
def solve_qe(a,b,c):
    delta = b**2 - 4*a*c
    if delta >= 0:
        ans1 = (-b + sqrt(delta))/(2*a)
        ans2 = (-b - sqrt(delta))/(2*a)
        print("The two solutions are " + str(ans1)
              + " and " + str(ans2))
    else:
        print("The equation has no real root")
```

# Conditional

**Syntax**

```
if <expr>:
  statement(s)
```

**Example**

```
>>> my_money = 1000
>>> if my_money > 0:
        print('Good')
'Good'
```

# Conditional

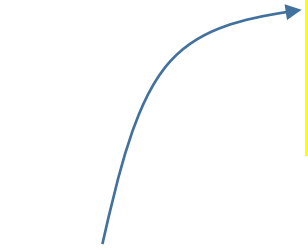
**Syntax**

```
if <expr>:
  statement(s)
```

**Example**

```
>>> my_money = 1000
>>> if my_money > 0:
        print('Good')
        print('Good')
        print('Good')

'Good'
'Good'
'Good'
```



# Conditional

**Syntax**

```
if <expr>:
    statement(s)
else:
    statement(s)
```

**Example**

```
>>> my_account = 1000
>>> if my_account > 0:
        print('rich')
else:
        print('broke')
'rich'
```

# Conditional (Nested)

**Syntax**

```
if <expr>:
    if <expr>:
        statement(s)
```

**Example**

```
a = 4
if a < 10:
    if a < 1:
        print('Here')
```

Print nothing

# Conditional

**Syntax**

```
if <expr>:
    statement(s)
else:
    statement(s)
```

**Example**

```
>>> my_account = 1000
>>> if my_account < 0:
        print('poor')
    else:
        if my_account > 1:
            print('v rich')

v rich
```

Clumsy

# Conditional

**Syntax**

```
if <expr>:
    statement(s)
elif <expr>:
    statements(s)
else:
    statement(s)
```

**Example**

```
>>> a = -3
>>> if a > 0:
        print('yes')
elif a == 0:
        print('no')
else:
        print('huh')
'huh'
```

# Conditional

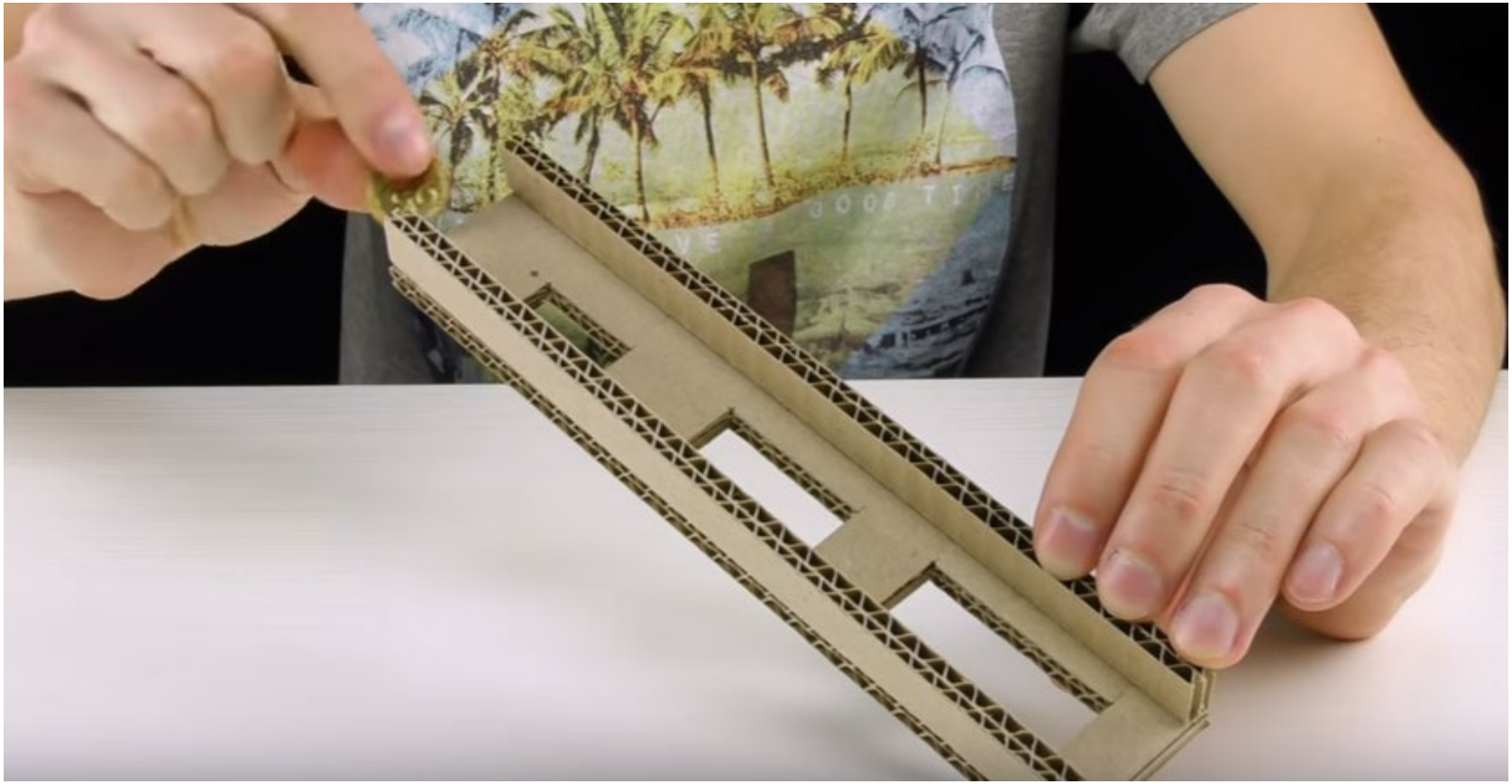
**Syntax**

```
if <expr>:
    statement(s)
elif <expr>:
    statements(s)
elif <expr>:
    statements(s)
else:
    statement(s)
```

Can be many

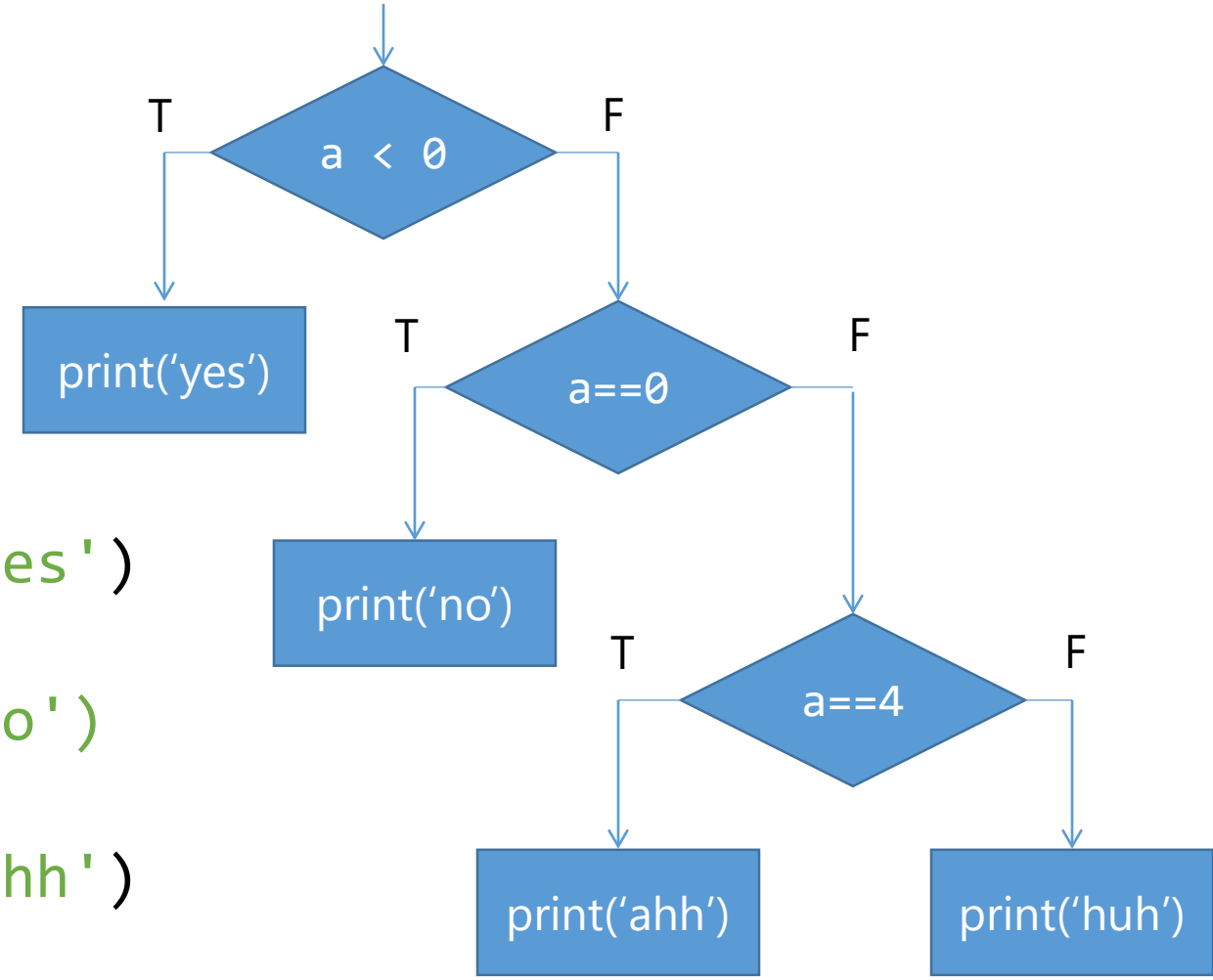
**Example**

```
>>> a = 4
>>> if a > 0:
        print('yes')
elif a == 0:
        print('no')
elif a == 4:
        print('ahh')
else:
        print('huh')
'yes'
```



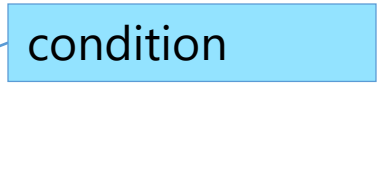
e.g.

```
>>> a = 4
>>> if a < 0:
        print('yes')
elif a == 0:
        print('no')
elif a == 4:
        print('ahh')
else:
        print('huh')
'ahh'
```



# Homework: Figure out ALL conditions

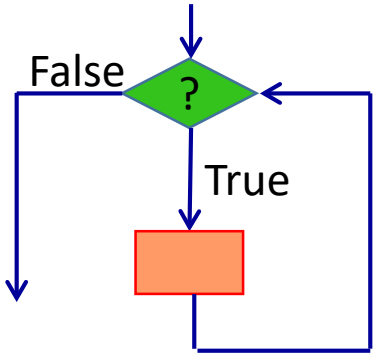
```
def solve_qe(a,b,c):
    delta = b**2 - 4*a*c
    if delta >= 0:
        ans1 = (-b + sqrt(delta))/(2*a)
        ans2 = (-b - sqrt(delta))/(2*a)
        print("The two solutions are " + str(ans1)
              + " and " + str(ans2))
    else:
        print("The equation has no real root")
```



# Repetition


I will not argue with idiots on YouTube.
I will not argue with idiots on YouTube.
I will not argue with idiots on YouTube.
I will not argue with idiots on YouTube.
I will not argue with idiots on YouTube.

# Control Structure: Repetition




- While (a condition)
  - ➢ Do something

- For example

```
While (I am hungry)
  Eat a bun
```

- Again, can be more than one single instruction

```
While(I have money in bank)
  Take money out from bank
  Eat an expensive meal
  While(I have money in my wallet)
    Go Shopping
```

# Iteration

the act of repeating a process with the aim of approaching a desired goal, target or result.

*- Wikipedia*

# Three Types of Loops

- Must run exactly N times
- Run any number of times
- Run at most N times
  - Check all True (or check all False)
  - Find any True (or False)

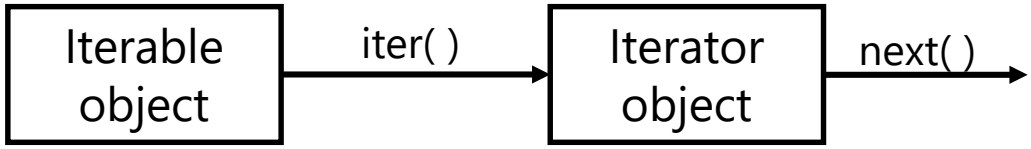
# Three Types of Loops

- **Must run exactly N times**
- Run any number of times
- Run at most N times
  - Check all True (or check all False)
  - Find any True (or False)

# For loop

- Uses iterable objects to repeatedly execute a series of tasks
- Number of repetitions are equal to number of items provided by iterable object
- Let us first look at iterable objects

# Iterable and Iterators




- Can be looped over
- Any object with __iter__ method is iterable
- Builtin Iterables:
  - range, strings, lists, tuples
- User-defined iterable objects
  - Generator functions

- Repeated calls of next() method return next item from iterator object
- If no further items in iterator, next( ) method raises StopIteration exception

# Iterables

- This week:
  - range()  - builtin iterable

- Subsequently
  - strings
  - user-defined iterators
  - lists/tuples/dictionary

# Builtin Iterable: range()

- range(start, stop, step)
  - ➢ Generate sequence of numbers from *start* (inclusive) to *stop* (exclusive), incremented by *step*
  - ➢ start and step are optional arguments
    - Default Values:
      - start = 0
      - step = 1

# Repetition Flow Control: “For”

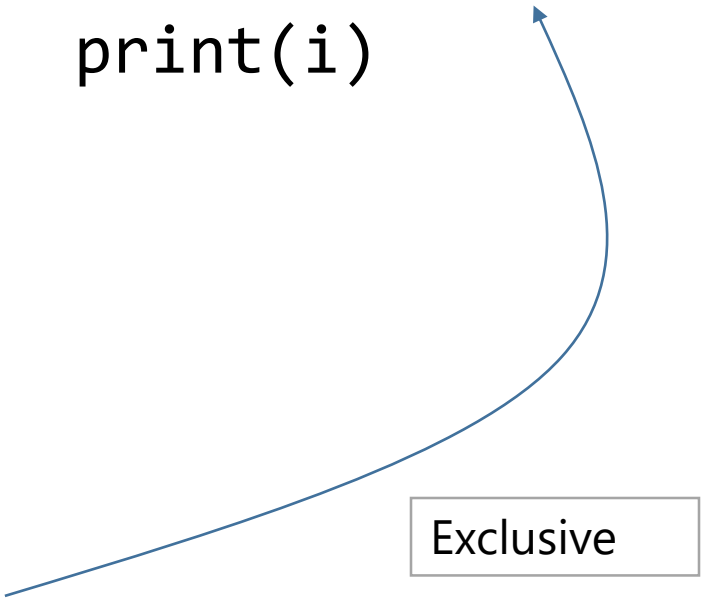
**Syntax**

```
for i in range(n,m):
    statement(s)
```

**Example**

```
for i in range(0,5):
    print(i)
```

```
0
1
2
3
4
```



# Repetition Flow Control: "For"

**Example**

```
for i in range(0,5):
      print(i)
```

```
0
1
2
3
4
```

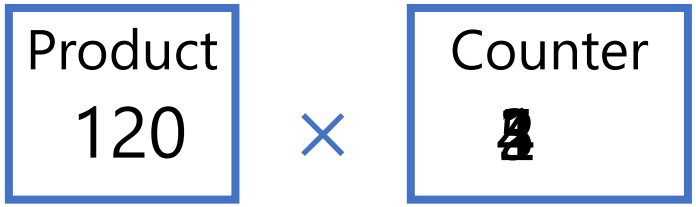
**Interpreted as**

```
i=0
print(i)
i=1
print(i)
i=2
print(i)
i=3
print(i)
i=4
print(i)
```

# Iterative Factorial

## Idea

- Start with 1, multiply by 2, multiply by 3, … , multiply by n.
- $n! = 1 \times 2 \times 3 \cdots \times n$



# Iterative Factorial

- $n! = 1 \times 2 \times 3 \cdots \times n$
- **Computationally**
- Starting:
- product = 1
- counter = 1
- Iterative (repeating) step:
- product ← product × counter
- counter ← counter + 1
- End:
- product contains the result

# Computing Factorial

- $n! = 1 \times 2 \times 3 \cdots \times n$
- Factorial rule:
- product ← product × counter
- counter ← counter + 1

non-inclusive.
Up to n.

```
def factorial(n):
    product = 1
    for counter in range(2, n+1):
        product = product * counter
    return product


factorial(6)
```

| product | counter |
|---|---|
| 1 | 2 |
| 1x2 = 2 | 3 |
| 1x2x3=6 | 4 |
| 1x2x3x4=24 | 5 |
| 120 | 6 |
| 720 | 7 |

# for loop

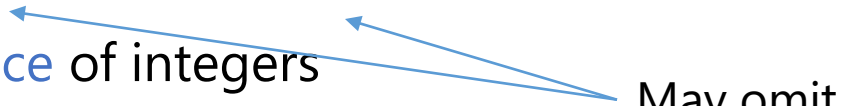
- for <var> in <sequence>:
- <body>

- sequence
  - a sequence of values
- var
  - variable that take each value in the sequence
- body
  - statement(s) that will be evaluated for each value in the sequence

# range function

- range([start,] stop[, step])

- creates a sequence of integers
  - from start (inclusive) to stop (non-inclusive)
  - incremented by step



# Examples

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

for i in range(3, 10):
    print(i)

for i in range(3, 10, 4):
    print(i)
```

# Example

Flipping coins

# Flipping a coin

- A coin is “fair” if the probably of getting a head is equal to a tail
  - P(head) == P(tail) == 0.5
- How to test a coin is fair?
- Flip 1000 times!

# Write a Pseudo Code for the Experiment

- I will flip a coin 1000 times and FOR EACH FLIP
  - I will record how many times I had flipped
  - If it is a head, I will record the number of heads

What you repeat for EACH time

# Flipping Coins

```
import random

def flipCoins():
    print('I will flip a coin 1000 times. ')
    print('Guess how many times it will come up heads. ')

    heads = 0
    for flip in range(0,1000):
        if random.randint(0, 1) == 1:
            heads = heads + 1
```

Randomly generate either 0 or 1

# while loop

- while <expression>:
- <body>

- expression
  - Predicate (condition) to stay within the loop
- body
  - Statement(s) that will be evaluated if predicate is True

# Repetition (Infinite)

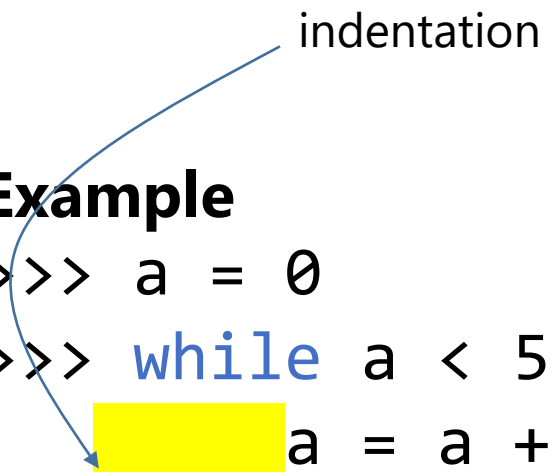
```
while True:
    print('Ah...')
```

```
a = 0
while a > 0:
    a = a + 1
    print(a)
```

# Repetition

**Syntax**

```
while <expr>:
    statement(s)
```




**Example**

```
>>> a = 0
>>> while a < 5:
        a = a + 1
        print(a)
1
2
3
4
5
```

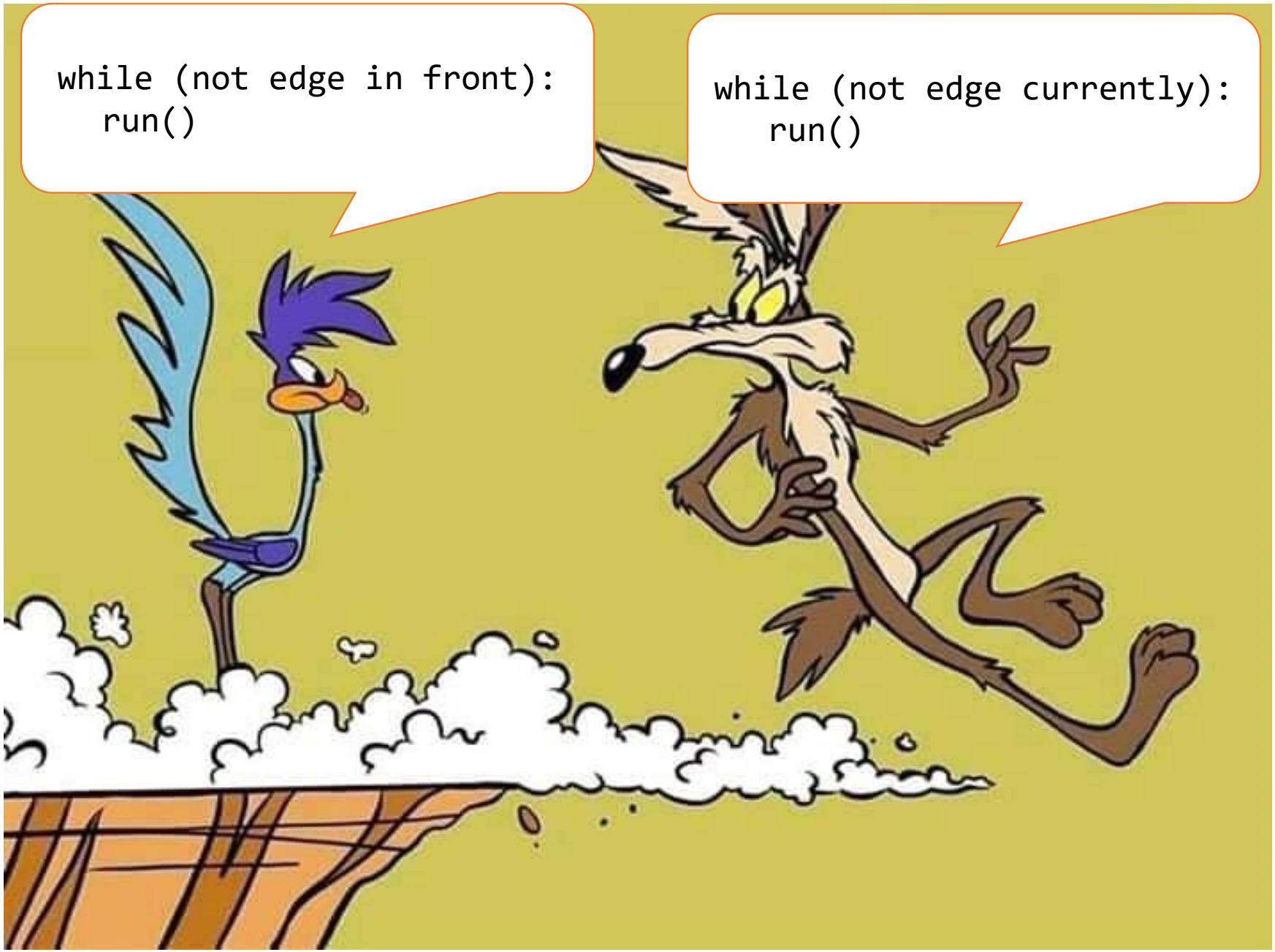
# Another Iterative process

```
def factorial(n):
    product, counter = 1, 1
    while counter <= n:
        product = (product *
                   counter)
        counter = counter + 1
    return product

factorial(6)
```

| product | counter |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 3 |
| 6 | 4 |
| 24 | 5 |
| 120 | 6 |
| 720 | 7 |

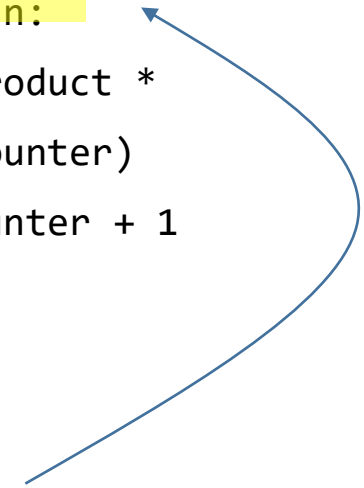
counter > n (7 > 6)
return product (720)


while (not edge in front):
run()
while (not edge currently):
run()

# Another Iterative process

```
def factorial(n):
    product, counter = 1, 1
    while counter <= n:
        product = (product *
                   counter)
        counter = counter + 1
    return product

factorial(6)
```



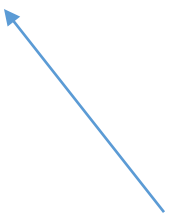
"<=" or "<" ?

| product | counter |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 3 |
| 6 | 4 |
| 24 | 5 |
| 120 | 6 |
| 720 | 7 |

counter > n (7 > 6)
return product (720)

```
import random

def flipCoins():
    print('I will flip a coin 1000 times. ')
    print('Guess how many times it will come up heads. ')
    flips = 0
    heads = 0
    while flips < 1000:
        if random.randint(0, 1) == 1:
            heads = heads + 1
        flips = flips + 1
```



Randomly generate either 0 or 1

# Repetition (nested)

indentation

**Syntax**

```
while <expr>:
    while <expr>:

statement(s)
```

**Example**

```
def nestedWhile():
    i = 0
    while i < 5:
        i += 1
        j = 0
        while j < 3:
            j += 1
            print ('#' * j)
```

```
#
##
###
#
##
###
#
##
###
#
##
###
#
##
###
```

# Repetition, a Very Common Pattern

**9 out of 10 times you will do**

```
>>> a = 0
>>> while a < N:
        a = a + 1
        do something
```

**For loop**

```
for i in range(0,N):
    do something
```

# Three Types of Loops

- Must run exactly N times
- **Run any number of times**
- Run at most N times
  - Check all True (or check all False)
  - Find any True (or False)

# Sum Some Numbers

```
Please enter a number or type 'bye' to sum:
12
Please enter a number or type 'bye' to sum:
99
Please enter a number or type 'bye' to sum:
123
Please enter a number or type 'bye' to sum:
2
Please enter a number or type 'bye' to sum:
bye
The sum of all numbers is 236
```

# Sum Some Numbers

- You do not know how many numbers will the user enter

```
def sumNumbers():
    sumSoFar = 0
    print('Please enter a number or type \'bye\' to sum:')
    num = input()
    while num != 'bye':
        sumSoFar += int(num)
        print('Please enter a number or type \'bye\' to sum:')
        num = input()
    print(f'The sum of all numbers is {sumSoFar}')

sumNumbers()
```

# Why do we need to repeat these?

```
def sumNumbers():
    sumSoFar = 0
    print('Please enter a number or type \'bye\' to sum:')
    num = input()
    while num != 'bye':
        sumSoFar += int(num)
        print('Please enter a number or type \'bye\' to sum:')
        num = input()
    print(f'The sum of all numbers is {sumSoFar}')

sumNumbers()
```

# Loop Terminating Condition

- Must run exactly N times
- **Run any number of times**
- Run at most N times
  - Check all True (or check all False)
  - Find any True (or False)

- If we do not know how many times do we need, when do we know we finish looping?

# Loop Terminating Condition

- When will the loop terminate?

```
def sumNumbers():
    sumSoFar = 0
    print('Please enter a number or type \'bye\' to sum:')
    num = input()
    while num != 'bye':
        sumSoFar += int(num)
        print('Please enter a number or type \'bye\' to sum:')
        num = input()
    print(f'The sum of all numbers is {sumSoFar}')
```

- When
  - num == ‘bye’? Or
  - num!= ‘bye’ ?

# Loop Terminating Condition

- When will the loop terminate?

```
def sumNumbers():
    sumSoFar = 0
    print('Please enter a number or type \'bye\' to sum:')
    num = input()
    while num != 'bye':
        sumSoFar += int(num)
        print('Please enter a number or type \'bye\' to sum:')
        num = input()
    print(f'The sum of all numbers is {sumSoFar}')
```

- The loop body will keep repeating if the condition is true
- The you break the loop if the condition is not true anymore

# Three Types of Loops

- Must run exactly N times
- Run any number of times
- **Run at most N times**
  - **Check all True** (or check all False)
  - Find any True (or False)

# Check if a String is all Alphabets

- Given a string, example, `'abc123'`
- Check if all the characters are alphabets
  - Return True or False

- In real life, how do you check?

- For example, if you are the teacher with a lot of test scripts, how do you check if "all are marked"?

# Goal: All are Alphabet

- I just need one of the answers:
  - Yes: if all are alphabet
  - No: if there exists one not alphabet
- Combining
  - You check the character one-by-one
  - If the current one is alphabet, do nothing, check the next
    - Else return “No”!
  - Until finishing all character all checked, return “Yes”

Which line you repeat a lot of times?

# Goal: All are Alphabet

- I just need one of the answers:
  - Yes: if all are alphabet
  - No: if there exists one not alphabet
- Combining
  - You check the character one-by-one
    - If the current one is alphabet, do nothing, check the next
      - Else return “No”!
  - Until finishing all character all checked, return “Yes”

In Python, you indent the statements needed to be loop

# Goal: All are Alphabets

- Combining
  - ➢ You check the character one-by-one
    - If the current one is NOT alphabet, return “No”!
  - ➢ Until finishing all character all checked, return “Yes”

```
def checkAllAlpha(string):
    l = len(string)
    for i in range(l):
        if not isAlphabet(string[i]):
            return False
    return True
```

# Goal: All are Alphabets

- Combining
  - ➢ You check the character one-by-one
    - If the current one is NOT alphabet, return “No”!
  - ➢ Until finishing all character all checked, return “Yes”

```
def checkAllAlpha(string):
    l = len(string)
    for i in range(l):
        if not isAlphabet(string[i]):
            return False
    return True
```

# Goal: All are Alphabets

- Combining
  - ➢ You check the character one-by-one
    - If the current one is NOT alphabet, return “No”!
  - ➢ Until finishing all character all checked, return “Yes”

```
def checkAllAlpha(string):
    l = len(string)
    for i in range(l):
        if not isAlphabet(string[i]):
            return False
    return True
```

# Goal: All are Alphabets

- Combining
  - ➢ You check the character one-by-one
    - If the current one is NOT alphabet, return “No”!
  - ➢ Until finishing all character all checked, return “Yes”

```
def checkAllAlpha(string):
    l = len(string)
    for i in range(l):
        if not isAlphabet(string[i]):
            return False
    return True
```

# How many times?

- How many times we reach this line if len(string) = N?
  - ➢ You check the character one-by-one
    - If the current one is NOT alphabet, return “No”!
  - ➢ Until finishing all character all checked, return “Yes”

```
def checkAllAlpha(string):
    l = len(string)
    for i in range(l):
        if not isAlphabet(string[i]):
            return False
    return True
```

- Worst case: N times
- But maybe less than N

# Provided that we have a function

- To check if a character is an alphabet

```
def isAlphabet(s):
    if s >= 'a' and s <= 'z':
        return True
    if s >= 'A' and s <= 'Z':
        return True
    return False
```

# Three Types of Loops

- Must run exactly N times
- Run any number of times
- **Run at most N times**
  - Check all True **(or check all False)**
  - Find any True (or False)

- Check all True similar to check all False
  - E.g. check if all characters are NOT alphabet?

```
def checkAllNotAlpha(string):
    l = len(string)
    for i in range(l):
        if isAlphabet(string[i]):
            return False
    return True
```

# Three Types of Loops

- Must run exactly N times
- Run any number of times
- **Run at most N times**
  - Check all True (or check all False**)**
  - **Find any True (or False)**

- Check any True?
  - Return the reverse of “check all False”

```
def checkAnyAlpha(string):
    return not checkAllNotAlpha(string)
```

# “For” vs “While”

- When to use “for” and when to use “while”?
- “For”
  - You know how many times before hand
  - Namely, anything in the body of the loop will NOT change the number of times you repeat the loop
  - E.g. printing out all the data in a spreadsheet
- “While”
  - You may not know how many times you need to repeat
  - The number of times is depended on the “condition”, in which, may change unpredictably inside the loop
  - E.g. while the player haven’t guess the right answer, keep guessing

# Lastly: break & continue

```
for j in range(10):
    print(j)
    if j == 3:
        break
print("done")
```

Jump

```
0
1
2
3
done
```

Break out of loop

```
for j in range(10):
    if j % 2 == 0:
        continue
    print(j)
print("done")
```

Jump

```
1
3
5
7
9
done
```

Continue with next value

# Let’s play a game

```
>>> guessANum()
I have a number in mind between 0 and 99
Guess a number: 50
Too big
Guess a number: 25
Too big
Guess a number: 12
Too big
Guess a number: 6
Too small
Guess a number: 9
Too big
Guess a number: 7
Bingo!!!
>>>
```

# guessANum.py

```
import random

def guessANum():
    secret = random.randint(0,99)    # 0 <= secret <= 99
    guess = -1
    print('I have a number in mind between 0 and 99')
    while guess != secret:
        guess = int(input('Guess a number: '))
        if guess == secret:
            print('Bingo!!! You got it! ')
        elif guess < secret:
            print('Your number is too small')
        else:
            print('Your number is too big')

guessANum()
```

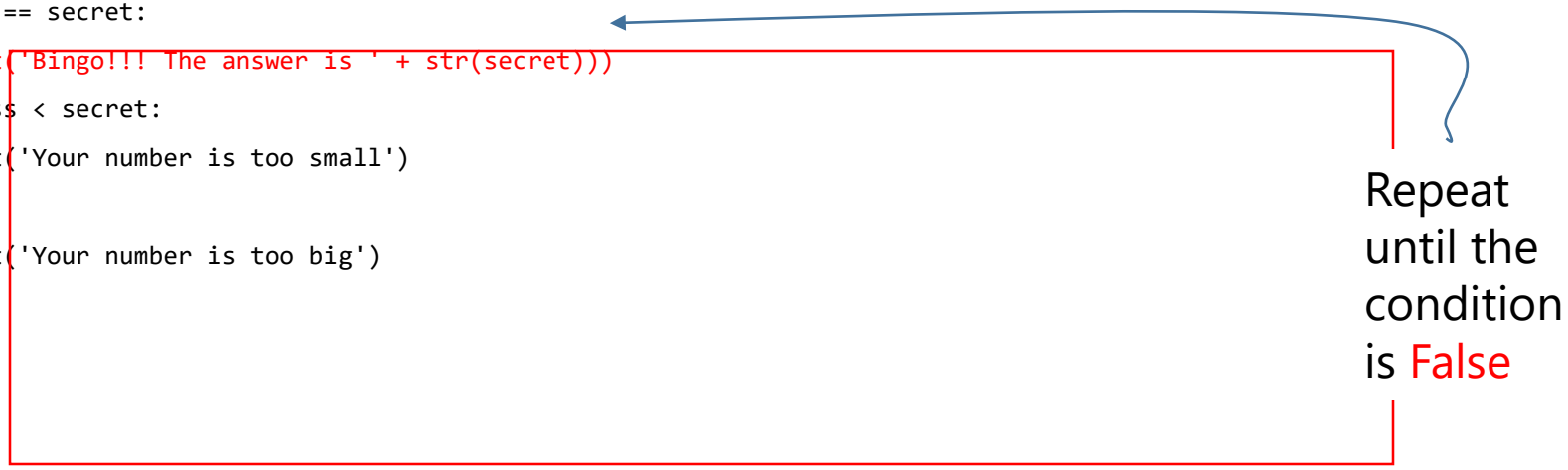
Repeat until the condition is False

# guessANum.py

```
import random

def guessANum():
    secret = random.randint(0,99)    # 0 <= secret <= 99
    guess = -1
    print('I have a number in mind between 0 and 99')
    while guess != secret:
        guess = int(input('Guess a number: '))
        if guess == secret:
            print('Bingo!!! The answer is ' + str(secret))
        elif guess < secret:
            print('Your number is too small')
        else:
            print('Your number is too big')


guessANum()
```



# How to write a love letter in Python

```
def show_my_love():
    everything = True
    you = everything
    my_mind = you
    while(my_mind == True):
        print('I love you')
```

# How to write a love letter in Python

```
def show_my_love():
    everything = True  #Everything I say is true
    you = everything   #You are everything to me
    my_mind = you      #All my mind is filled with you

    # No 'if' in my love because it's unconditioal

    while(my_mind == True): # My love is eternal
        print('I love you')

    # And there is no 'return' in my love
    # because I do not expect any
```

# Tips

- A “while” or “if” block starts with a colon “:”
- Remember
  - When there is a colon, there are indentations
  - When there are indentations, before these there is a colon

- The inclusive/exclusive range is a pain