FLOW CONTROL

Johan Van Bauwel
Johan.vanbauwel@thomasmore.be
A114

YES AND NO VALUES

- Boolean Values
- Comparison Operators
- Boolean Operators

BOOLEAN VALUES

- Only 2 values: True and False
- Used in expressions
- Can be stored in variables

COMPARISON OPERATORS

- Comparison Operators are used to test for True or False
- Comparison operators are used in logical statements to determine equality or difference between variables or values.

TABLE COMPARISON OPERATORS

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

EXAMPLE

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True
>>> 2 != 2
False
```

```
>>> 'hello' == 'hello'
True
>>> 'hello' == 'Hello'
False
>>> 'dog' != 'cat'
True
>>> True == True
True
>>> True != False
True
```

EXAMPLE

```
>>> 42 == '42'
False
```

- Integer 42 is not the same as string 42
- Float and integer values can be equal

```
>>> 42 == 42.0
True
```

EXAMPLE WITH VARIABLES

```
>>> eggCount = 42
>>> eggCount <= 42
True
>>> myAge = 29
>>> myAge >= 10
True
```

DIFFERENCE BETWEEN == AND =

- The == operator (equal to) asks whether two values are the same
- The = operator (assignment) puts the value on the right into the variable on the left
- To help remember which is which, notice that the == operator (equal to) consists of two characters, just like the != operator (not equal to) consists of two characters

BOOLEAN OPERATORS

- 3 operators:
  - and
  - or
  - not

AND-OPERATOR

Expression	Evaluates to...
`True and True`	`True`
`True and False`	`False`
`False and True`	`False`
`False and False`	`False`

```
>>> True and True
True
>>> True and False
False
```

K.H.Kempen and Lessius are joining up to become *more*.

OR-OPERATION

Expression	Evaluates to...
True or True	True
True or False	True
False or True	True
False or False	False

```
>>> False or True
True
>>> False or False
False
```

K.H.Kempen and Lessius are joining up to become *more*.

NOT-OPERATOR

Expression	Evaluates to...
`not True`	`False`
`not False`	`True`

```
>>> not True
False
>>> not not not not True
True
```

MIXING BOOLEAN AND COMPARISON OPERATORS

- Recall that the and, or, and not operators are called Boolean operators because they always operate on the Boolean values True and False

- While expressions like 4 < 5 aren't Boolean values, they are expressions that evaluate down to Boolean values.

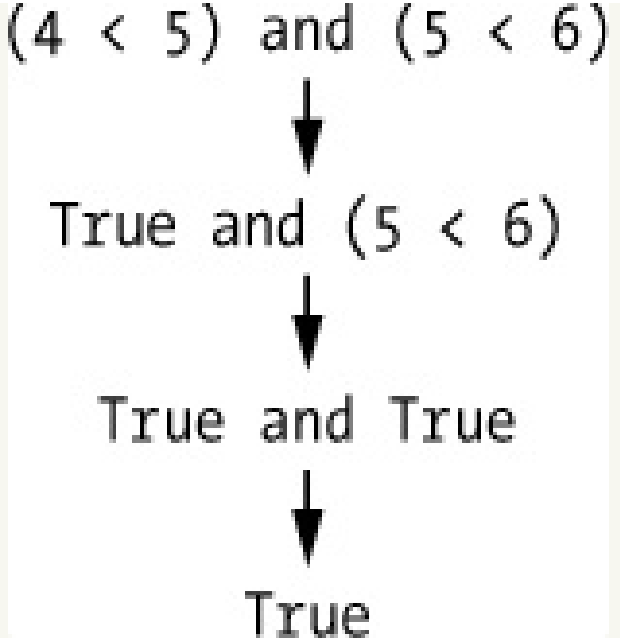
MIXING BOOLEAN AND COMPARISON OPERATORS EXAMPLE

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
```

- You can also use multiple operators

```
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

ORDER OF OPERATIONS



ELEMENTS OF FLOW CONTROL

- Conditions
  - Evaluate down to a Boolean value
  - Almost every flow control statement use a condition

ELEMENTS OF FLOW CONTROL

- Blocks of Code rules:
  - Blocks begin when the indentation increases
  - Blocks can contain other blocks
  - Blocks end when the indentation decreases to zero or to a containing block's indentation

```
if name == 'Mary':
    print('Hello Mary')
if password == 'swordfish':
    print('Access granted.')
else:
    print('Wrong password.')
```

K.H.Kempen and Lessius are joining up to become *more*.

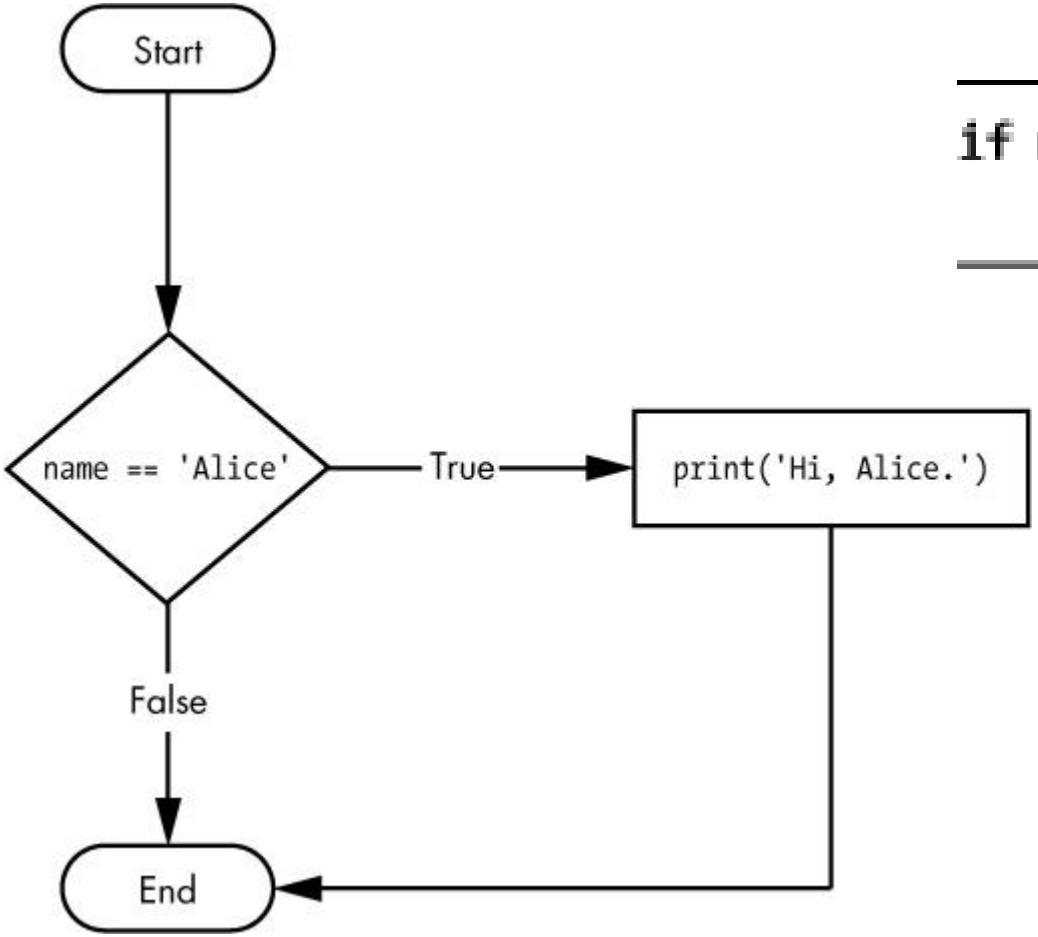
IF STATEMENTS

- The most common type of flow control statement is the if statement.
- An if statement's clause will execute if the statement's condition is True. The clause is skipped if the condition is False.
- In plain English, an if statement could be read as, "If this condition is true, execute the code in the clause"

IF STATEMENT

- In Python, an if statement consists of the following:
  - The *if* keyword
  - A condition (that is, an expression that evaluates to True or False)
  - A colon
  - Starting on the next line, an indented block of code (called the if clause)

IF STATEMENT




```
if name == 'Alice':
    print('Hi, Alice.')
```

K.H.Kempen and Lessius are joining up to become *more*.

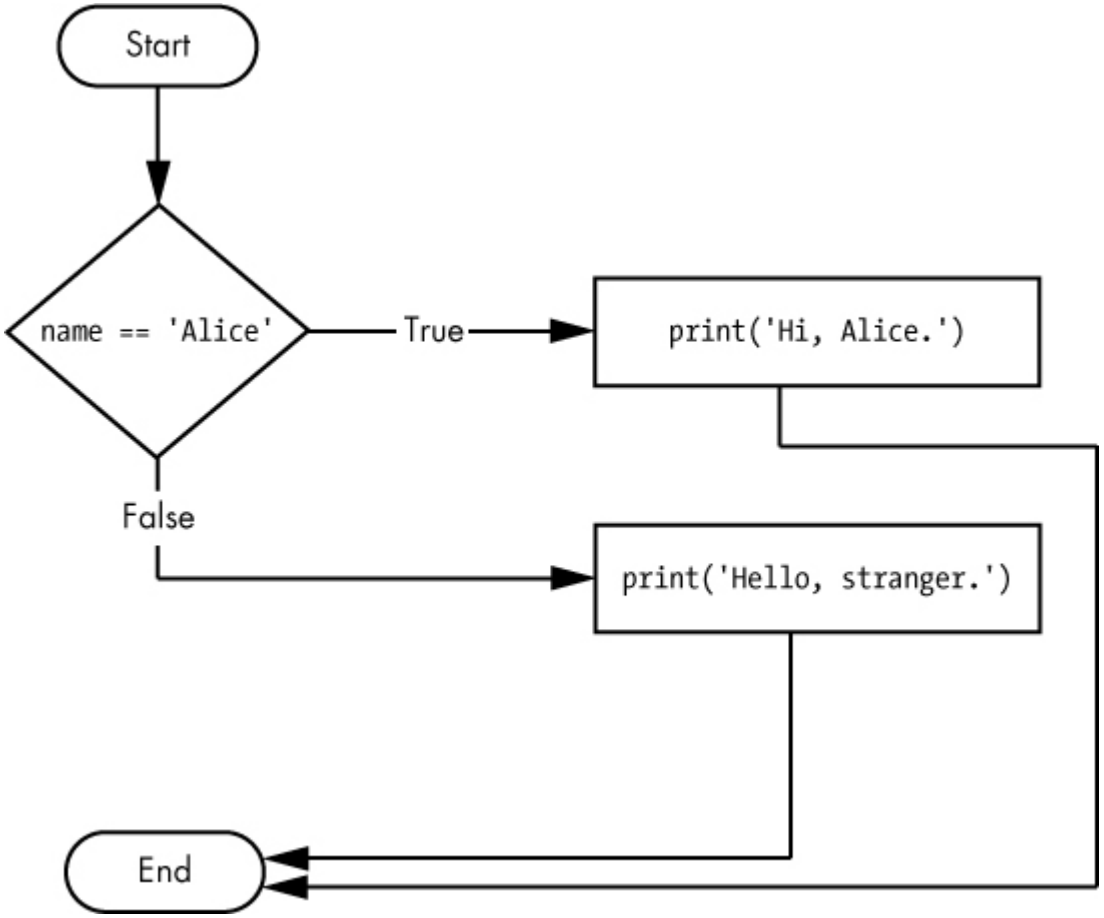
ELSE STATEMENT

- An if clause can optionally be followed by an else statement
- The else clause is executed only when the if statement's condition is False
- In plain English, an else statement could be read as, "If this condition is true, execute this code. Or else, execute that code."
- An else statement doesn't have a condition

ELSE STATEMENT

- Else statement in Python
  - The else keyword
  - A colon
  - Starting on the next line, an indented block of code (called the else clause)

ELSE STATEMENT




```
if name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

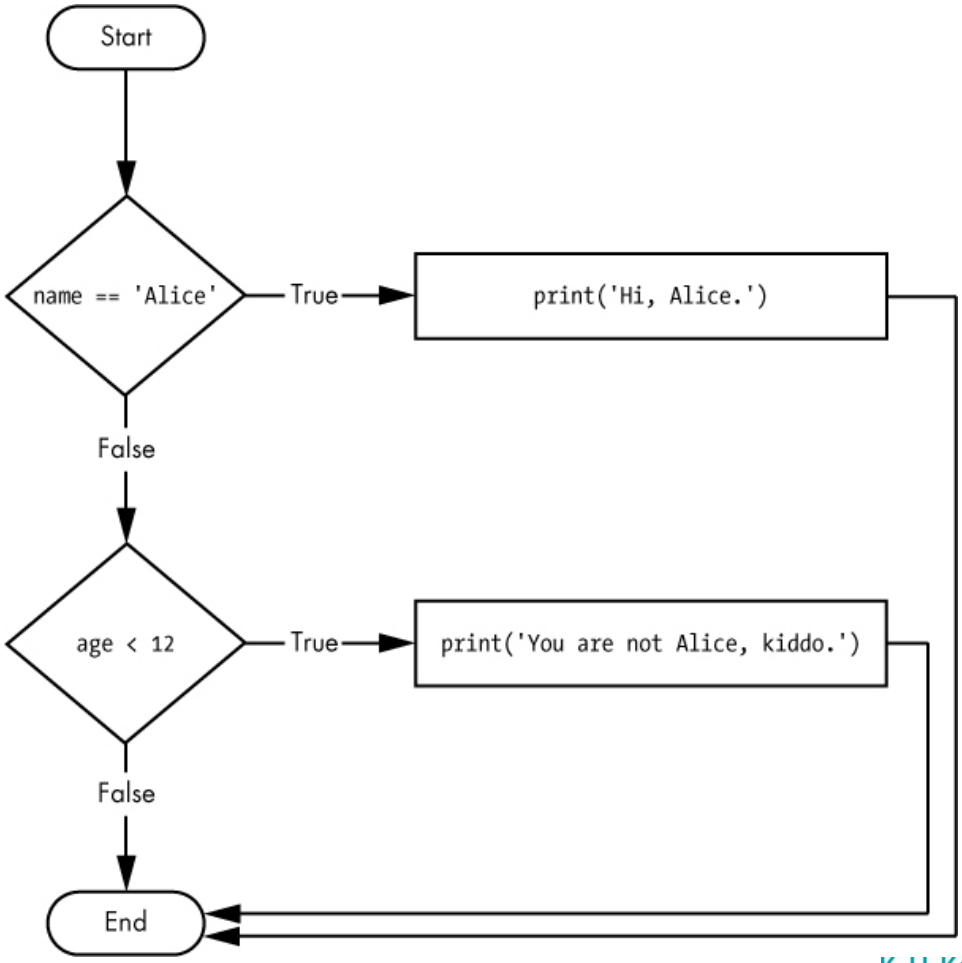
ELIF STATEMENT

- The elif statement is an “else if” statement that always follows an if or another elif statement
- It provides another condition that is checked only if any of the previous conditions were False

ELIF STATEMENT

- In code, an elif statement always consists of the following:
  - The elif keyword
  - A condition (that is, an expression that evaluates to True or False)
  - A colon
  - Starting on the next line, an indented block of code (called the elif clause)

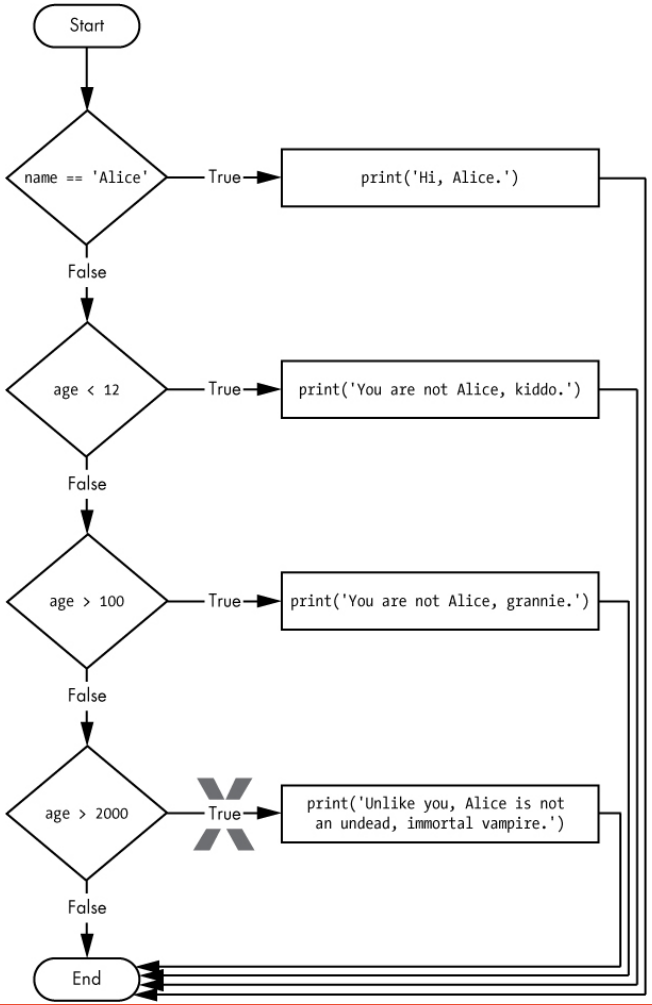
ELIF STATEMENT




```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
```

K.H.Kempen and Lessius are joining up to become *more*.

ELIF STATEMENT




```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 100:
    print('You are not Alice, grannie.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
```

K.H.Kempen and Lessius are joining up to become *more*.

ELIF STATEMENT

Important! When using elif, order matters!

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.')
```

Suppose age = 3000
Output:
'Unlike you, Alice is not an undead, immortal vampire'

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 100:
    print('You are not Alice, grannie.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
```

Suppose age = 3000
Output:
'You are not Alice, grannie'

WHILE LOOP STATEMENT

- You can make a block of code execute over and over again with a while statement.
- The code in a while clause will be executed as long as the while statement's condition is True

WHILE LOOP STATEMENT

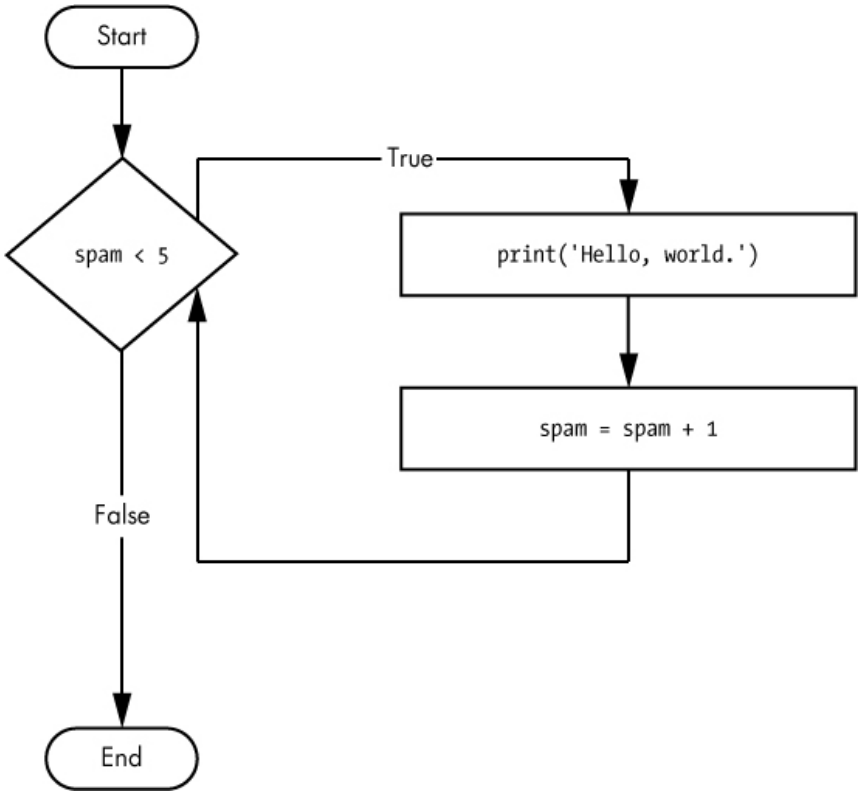
- In code, a while statement always consists of the following:
  - The while keyword
  - A condition (that is, an expression that evaluates to True or False
  - A colon
  - Starting on the next line, an indented block of code (called the while clause)

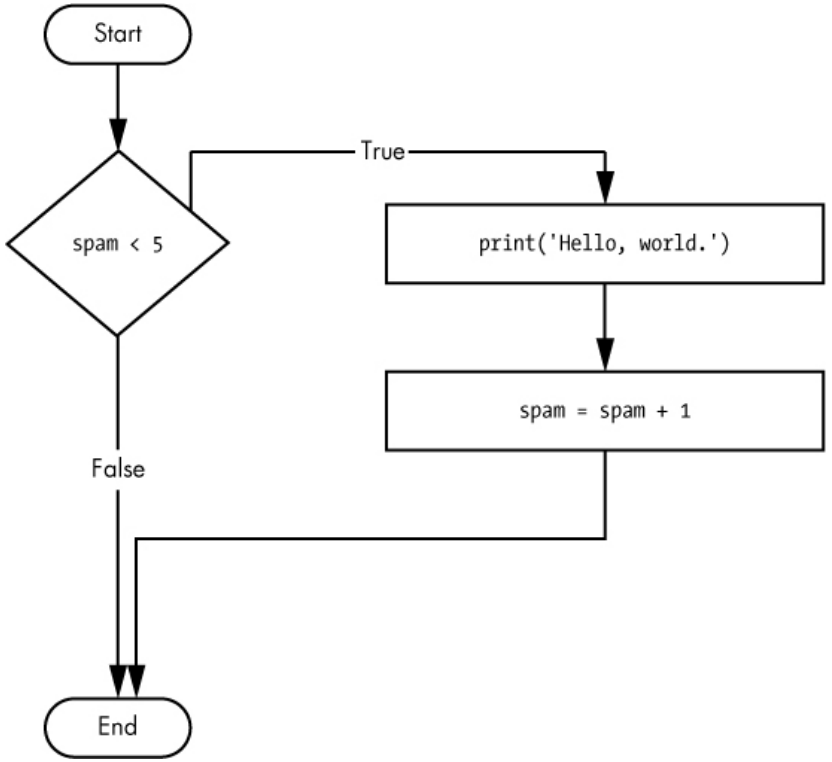
WHILE VS IF

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

WHILE VS IF



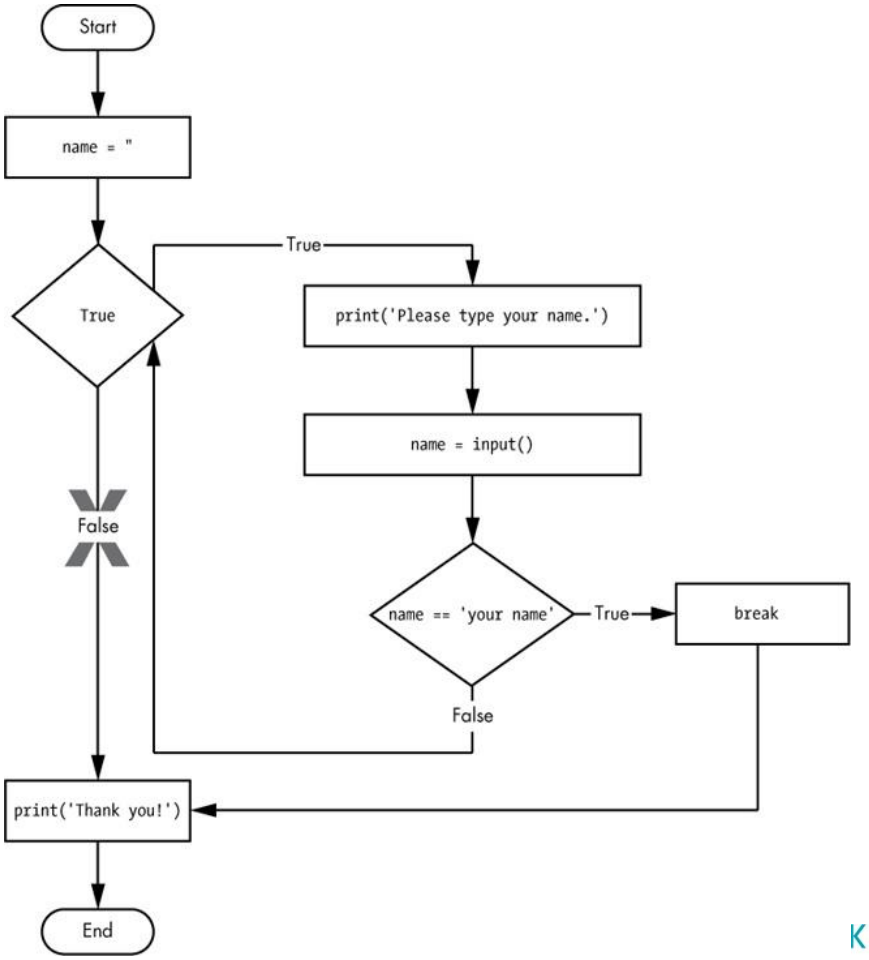


K.H.Kempen and Lessius are joining up to become *more*.

BREAK STATEMENT

- To break out of a while loop's clause early
- If the execution reaches a break statement, it **immediately exits the while loop's clause**
- In code, a break statement simply contains the break keyword

BREAK STATEMENT

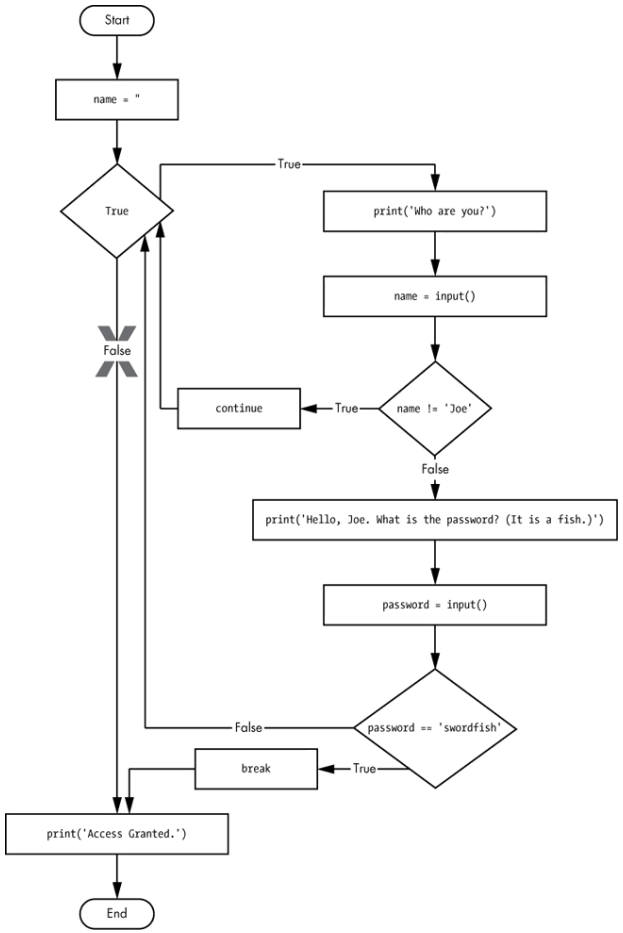



```
while True:
    print('Please type your name.')
    name = input()
    if name == 'your name':
        break
print('Thank you!')
```

CONTINUE STATEMENT

- Like break statements, continue statements are used inside loops.
- When the program execution reaches a continue statement, the program execution immediately **jumps back to the start of the loop and reevaluates the loop's condition**

CONTINUE STATEMENT




```
while True:
    print('Who are you?')
    name = input()
    if name != 'Joe':
        continue
    print('Hello, Joe. What is the password? (It is a fish.)')
    password = input()
    if password == 'swordfish':
        break
print('Access granted.')
```

K.H.Kempen and Lessius are joining up to become *more*.

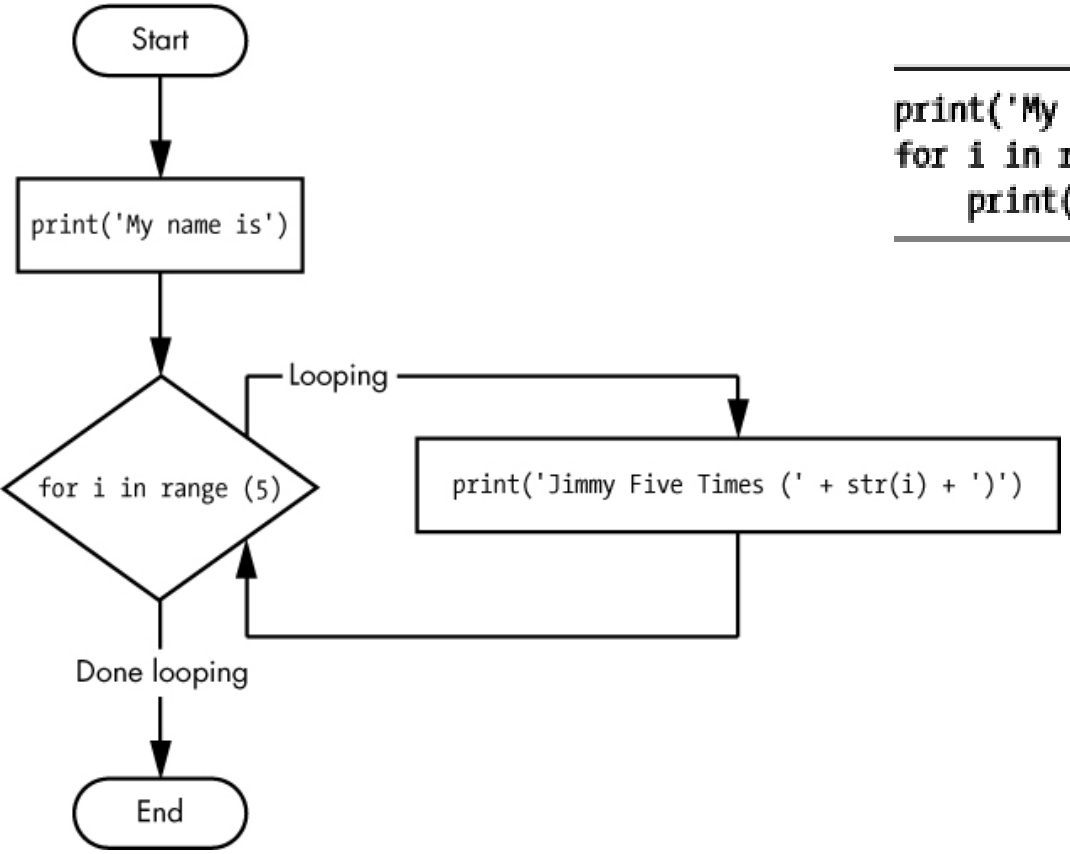
FOR LOOP AND RANGE() FUNCTION

- The while loop keeps looping while its condition is True but what if you want to *execute a block of code only a certain number of times*?
- You can do this with a **for loop** statement and the range() function

FOR LOOP AND RANGE() FUNCTION

- In code, a for statement looks something like for i in range(5): and always includes the following:
  - The for keyword
  - A variable name
  - The in keyword
  - A call to the range() method with up to three integers passed to it
  - A colon
  - Starting on the next line, an indented block of code (called the for clause)

FOR LOOP AND RANGE() FUNCTION




```
print('My name is')
for i in range(5):
    print('Jimmy Five Times (' + str(i) + ')')
```

```
My name is
Jimmy Five Times (0)
Jimmy Five Times (1)
Jimmy Five Times (2)
Jimmy Five Times (3)
Jimmy Five Times (4)
```

K.H.Kempen and Lessius are joining up to become *more*.

IMPORT RANDOM

```
import random
for i in range(5):
    print(random.randint(1, 10))
```

```
4
1
8
4
1
```

K.H.Kempen and Lessius are joining up to become *more*.

ENDING A PROGRAM EARLY WITH SYS.EXIT()

```
import sys

while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
        sys.exit()
    print('You typed ' + response + '.')
```

Q&A

- Questions?