Lecture 3 Conditional Statements

IDS 400
Programming for Data Science in Business

Lab Sessions

- Lab 2 is available on Blackboard.
- While lab submissions will not be graded, please feel free to reach out to the me or TA if detailed feedback is needed.

Piazza

- Everyone should have already signed up to Piazza by now
- Please read previous questions to avoid posting redundant questions

Assignment

Assignment 1 is available on Blackboard.

- Due: 06:00 pm Thursday Sept 14th.
- You are allowed to discuss with other students (up to three) offline. Please put all the names of students that you discussed with. However individual students must write their own solutions.
- Copying a program, or letting someone else copy your program, is a form of academic dishonesty. Any referred material must be cited properly.
  - E.g., append the links of the resources you used online at the end of your submission.
- Maximally leverage Piazza to benefit other students by your questions and answers.

Tentative schedule

Date	Lecture Number	Topics
08/24	Lecture 1	Introduction
08/31	Lecture 2	Basic
09/07	Lecture 3	Condition
09/14	Lecture 4	Loop
09/21	Lecture 5	String + **Quiz 1** → Online
09/28	Lecture 6	Type
10/05	Lecture 7	Function
10/12	Lecture 8	File + **Quiz 2** → Online
10/19	Lecture 9	Pandas
10/26	Lecture 10	Numpy
11/02	Lecture 11	Machine Learning
11/09	Lecture 12	Visualization
11/16	Lecture 13	Web Scraping & Deep Learning
11/23	*Thanksgiving*	*No lecture*
11/30	**Final presentation**	In class presentation
12/05	**Project submission due**	

Logical operators

- Logical operators are used to combine conditional statements.

Assume variable a has value ***True*** and b holds ***False***.

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is False
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is True

```python
x = True
y = False

print('x and y is',x and y)

print('x or y is',x or y)

print('not x is',not x)
```

```
x and y is False
x or y is True
not x is False
```

Logical operator

- a = 10. What is the value (True or False) of ‘not a’?

Logical operator

- a = 10. What is the value (True or False) of 'not a'?
  - For numerical types like integers and floating-points, **zero** values are **False** and **non-zero** values are **True**.
  - **'not a'** for strings/lists/tuples/dictionaries, or **space between quotes** are **False** and **empty objects (zero length)** are **True**.

```
a = 10
not a
```

```
False
```

```
e = [1, 2, 3]
not e
```

```
False
```

```
c = ' '   # one space
not c
```

```
b = 0
not b
```

```
True
```

```
f = []
not f
```

```
True
```

```
d = ''   # empty string
not d
```

Logical operator

- a = 10. What is the value (True or False) of 'not a'?
  - For numerical types like integers and floating-points, **zero** values are **False** and **non-zero** values are **True**.
  - For strings/lists/tuples/dictionaries, **empty objects (such as space)** are **False** and **non-empty objects** are **True**.

```
a = 10
not a
```

```
False
```

```
e = [1, 2, 3]
not e
```

```
False
```

```
c = ' '   # one space
not c
```

```
False
```

```
b = 0
not b
```

```
True
```

```
f = []
not f
```

```
True
```

```
d = ''   # empty string
not d
```

```
True
```

Comparison operators

- Compare values on either sides of them and decide the relation among them.

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Discarded in Python 3

```python
x = 10
y = 20

# Output: x > y is False
print('x > y is',x>y)

# Output: x < y is True
print('x < y is',x<y)

# Output: x == y is False
print('x == y is',x==y)

# Output: x != y is True
print('x != y is',x!=y)

# Output: x >= y is False
print('x >= y is',x>=y)

# Output: x <= y is True
print('x <= y is',x<=y)
```

```
x > y is False
x < y is True
x == y is False
x != y is True
x >= y is False
x <= y is True
```

Identity operators

- **is** and **is not** are the identity operators in Python. They are used to check if two values (or variables) are *located on the same part of the memory*.
- Two variables that are equal does not imply that they are identical.

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

```
x1 = 5
y1 = 5

# Output: False
print(x1 is y1)
```

```
True
```

x1 and y1 are constants (i.e., integers) of the same values, so they are equal as well as identical.

```
x2 = 'Hello'
y2 = 'Hello'

# Output: True
print(x2 is y2)
```

```
True
```

Same is the case with x2 and y2 (strings).

```
x3 = [1,2,3]
y3 = [1,2,3]

# Output: False
print(x3 is y3)
```

```
False
```

But x3 and y3 are lists. They are equal but not identical. It is because the interpreter locates them separately in memory although they are equal.

Identity operators

```
x = ["apple", "banana"]
y = ["apple", "banana"]

print(x is y)
```

```
print(x == y)
```

```
z = x

print(z is x)
```

```
print(z is y)
```

Identity operators

```
x = ["apple", "banana"]
y = ["apple", "banana"]

print(x is y)
# False, because x is not the same object as y, even if they have the same content
```

```
False
```

```
print(x == y)
```

```
z = x

print(z is x)
```

```
print(z is y)
```

Identity operators

```
x = ["apple", "banana"]
y = ["apple", "banana"]

print(x is y)
# False, because x is not the same object as y, even if they have the same content
```

```
False
```

```
print(x == y)
# to demonstrate the difference betweeen "is" and "=="
# this comparison returns True because x is equal to y
```

```
True
```

```
z = x

print(z is x)
```

```
print(z is y)
```

Identity operators

```
x = ["apple", "banana"]
y = ["apple", "banana"]

print(x is y)
# False, because x is not the same object as y, even if they have the same content
```

```
False
```

```
print(x == y)
# to demonstrate the difference betweeen "is" and "=="
# this comparison returns True because x is equal to y
```

```
True
```

```
z = x

print(z is x)
# True, because z is the same object as x
```

```
True
```

```
print(z is y)
```

Identity operators

```
x = ["apple", "banana"]
y = ["apple", "banana"]

print(x is y)
# False, because x is not the same object as y, even if they have the same content
```

```
False
```

```
print(x == y)
# to demonstrate the difference betweeen "is" and "=="
# this comparison returns True because x is equal to y
```

```
True
```

```
z = x

print(z is x)
# True, because z is the same object as x
```

```
True
```

```
print(z is y)
# False, because z is not the same object as y, even if they have the same content
```

```
False
```

Identity operators

```python
# id() function returns the unique identity of an object
# If we relate this to C, then they are actually the memory address, here in Python it is the unique id.

x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print('x id is:', id(x))
# x id is: 2746693449664

print('y id is:', id(y))
# y id is: 2746693520064

print('z id is:', id(z))
# z id is: 2746693449664
```

```
x id is: 3205773939264
y id is: 3205773938816
z id is: 3205773939264
```

Operator precedence

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.
  - *Parentheses have the highest precedence* and can be used to force an expression to evaluate in the order you want.
  - *Exponentiation has the next highest precedence*.
  - Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence .
  - *Operators with the same precedence are evaluated from left to right*.

Operator precedence

- The operator precedence in Python is listed in the following table. It is in descending order (upper group has higher precedence than the lower ones).

Operators	Meaning
`()`	Parentheses
`**`	Exponent
`+x`, `-x`, `~x`	Unary plus, Unary minus, Bitwise NOT
`*`, `/`, `//`, `%`	Multiplication, Division, Floor division, Modulus
`+`, `-`	Addition, Subtraction
`<<`, `>>`	Bitwise shift operators
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==`, `!=`, `>`, `>=`, `<`, `<=`, `is`, `is not`, `in`, `not in`	Comparisons, Identity, Membership operators
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR

A simple example

```
>>> x = 6

>>> x = 4 * x ** (8 - x)

>>> print(x)
```

What is the output?

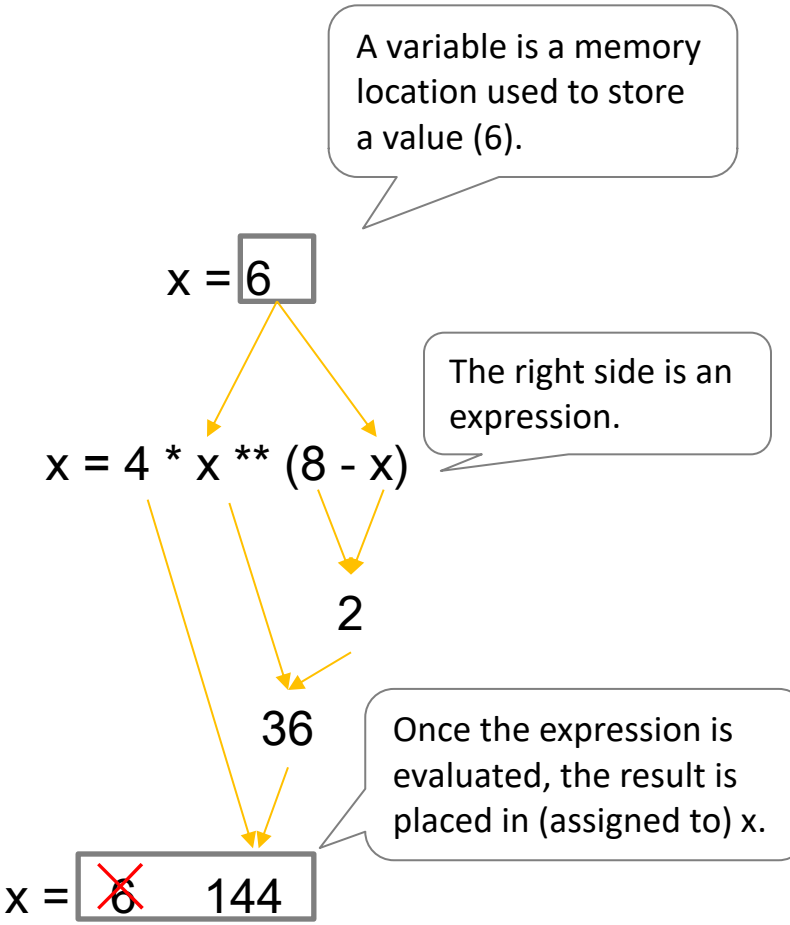
A simple example

```
>>> x = 6

>>> x = 4 * x ** (8 - x)

>>> print(x)
```

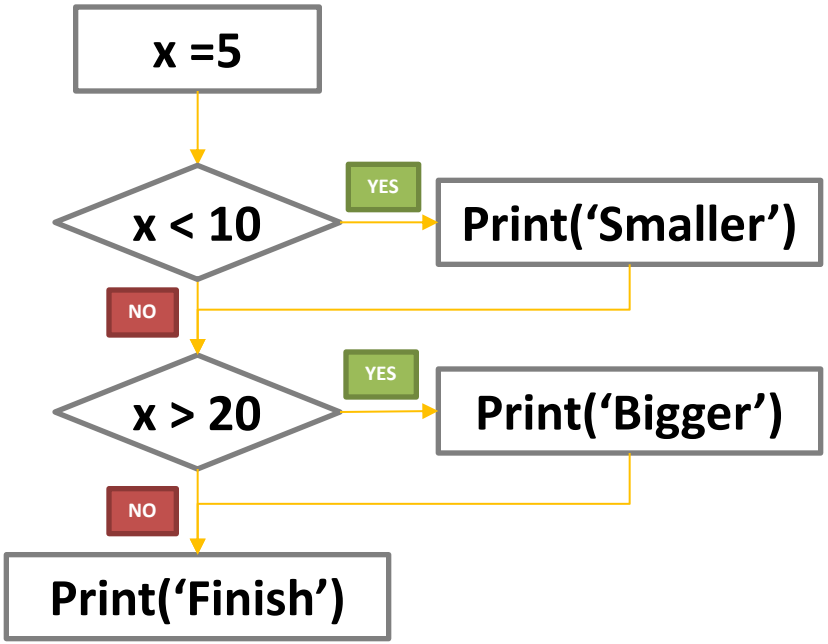
What is the output?



For This Class

- Condition/ Conditional Statement (*if* statement)

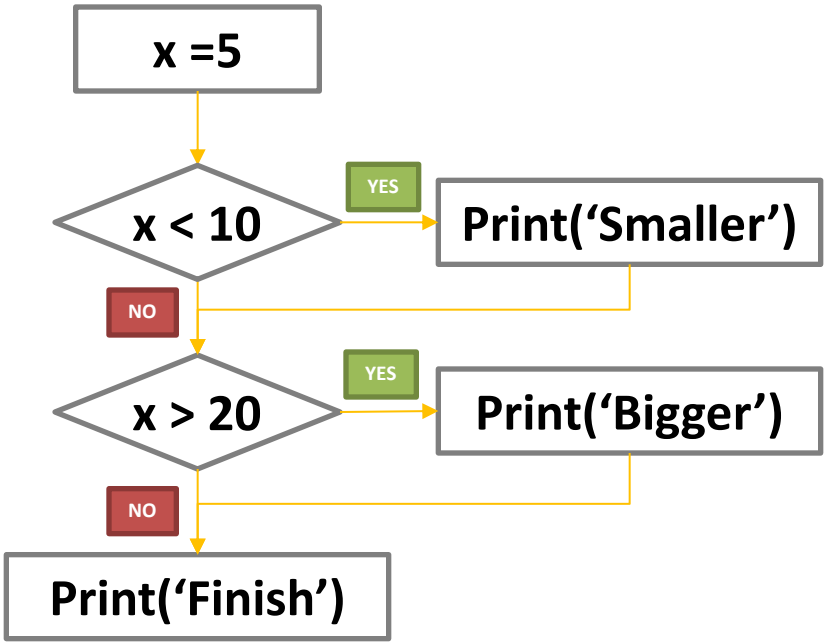
Conditional steps



Conditional steps

```
x =5
if x <10:
    print('Smaller')
if x >20:
    print('Bigger')
print('Finish')
```

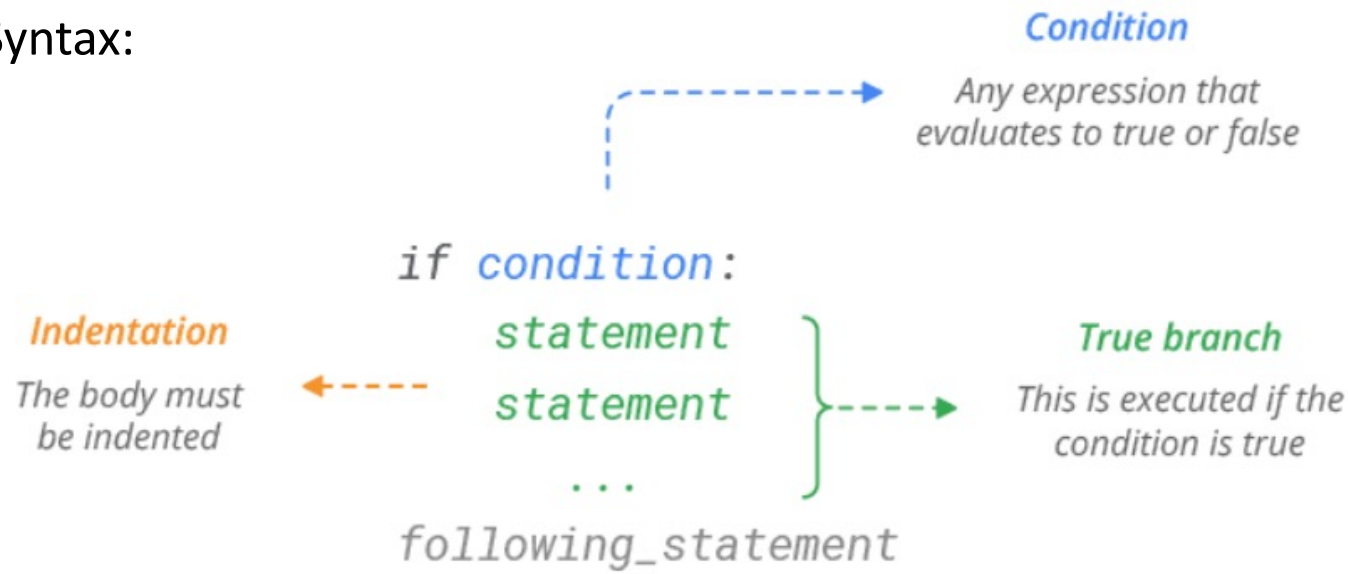
```
Smaller
Finish
```



In Python, If Statement is used for decision making.

If statement

Syntax:



If the expression is ***True***, statements <u>within the if statement body</u> will be executed, ***otherwise*** the entire "if statement" will be <u>ignored</u>.

If statement

- Basic examples

```
# mathematical expression
x, y = 7, 5
if x > y:
    print('x is greater')

# Prints x is greater
```

```
x is greater
```

```
# any non-zero value
if -3:
    print('True')

# Prints True
```

```
True
```

```
# nonempty container
L = ['red','green']
if L:
    print('True')

# Prints True
```

```
True
```

Indentation

- Indentation has **a special significance** in Python. It is used to define *a block of code* (often referred to as, a suite). Contiguous statements that are indented to the same level are considered as part of the same block.
- *if* statement without indentation raises syntax error.

```
x, y = 7, 5
if x > y:
print('x is greater')
# Triggers SyntaxError: expected an indented block
```

Indentation

- **Increase indent** after an if statement (after :).
- **Maintain indent** to indicate the scope of the block (which lines are affected by the *if*).
- **Reduce indent** back to the level of the if statement to indicate the end of the block.
- Blank lines are ignored - they do not affect indentation.
- Comments on a line by themselves are ignored with regard to indentation.

Indentation

- **Increase** / **maintain** indent after *if*
- **Decrease** to indicate end of block

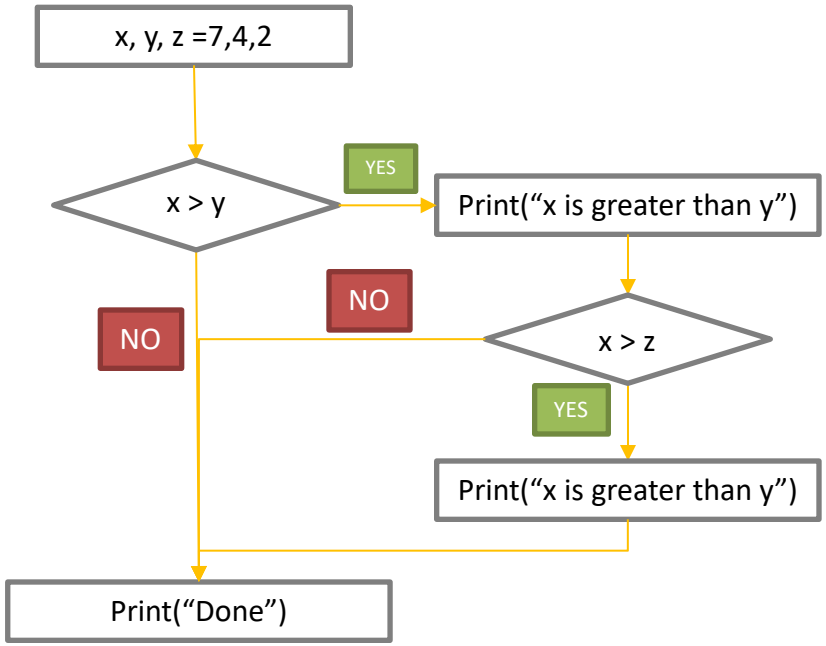
```
x, y = 7, 5
if x > y:
    print ('x is greater')
    print  ('Still greater')
print ('Done')
# Prints x is greater
```

Nested if statement

- You can nest statements within a code block to begin a new code block, as long as they follow their respective indentations.
- You can have ***if*** statements inside ***if*** statements, this is called ***nested if*** statements.

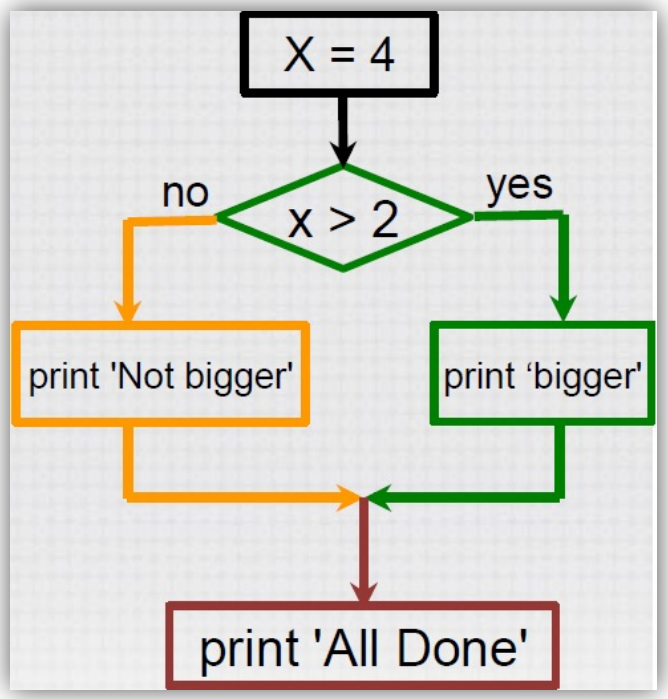
```
x, y, z = 7, 4, 2
if x > y:
    print ("x is greater than y")
    if x > z:
        print ("x is greater than y and z")
print ("Done")
```

```
x is greater than y
x is greater than y and z
Done
```



Else statement

- Sometimes we want to do one thing if a logical expression is true, and something else if the expression is false.
- It is like a fork in the road – we must choose one or the other path , but not both.



Else statement

- Use *else* statement to execute a block of Python code, if the condition is false. Syntax:

```
if condition:
    statement
    statement
        ...
else:
    statement
    statement
        ...
following_statement
```

True branch

This is executed if the condition is true

False branch

This is executed if the condition is false

Else statement

- The **else** keyword catches anything which isn't caught by the preceding conditions.
- If the expression is true, statements within the if statement body will be executed, <u>otherwise statements within else body will be executed</u>.

```
x, y = 7, 5
if x < y:
    print('y is greater')
else x > y:
    print('x is greater or equal')
```

Else statement

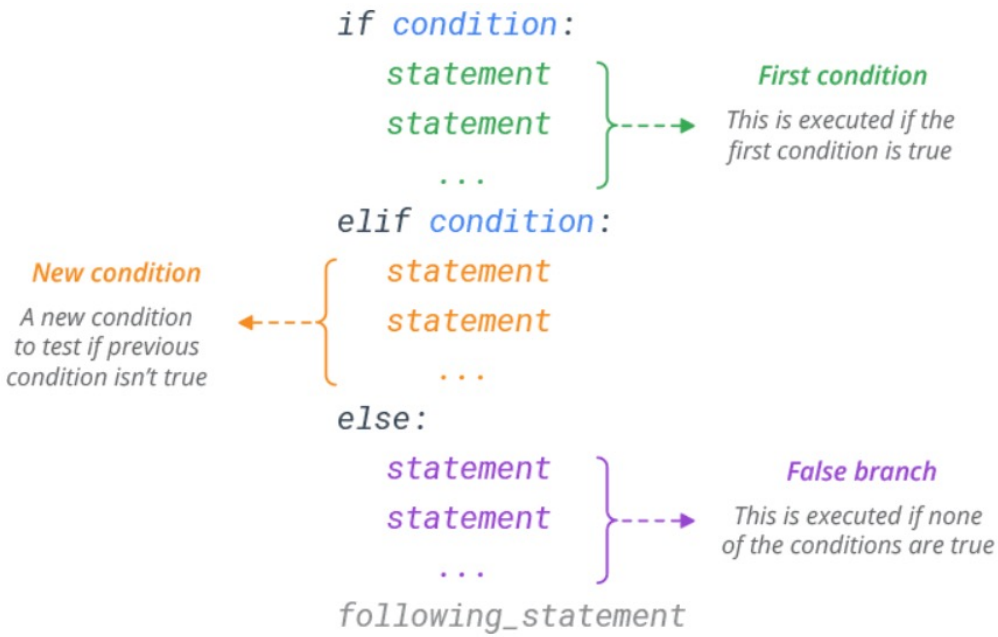
- The **else** keyword catches anything which isn't caught by the preceding conditions.
- If the expression is true, statements within the if statement body will be executed, <u>otherwise statements within else body will be executed</u>.

```
x, y = 7, 5
if x < y:
    print('y is greater')
else x > y:
    print('x is greater or equal')
```

```
  File "C:\Users\Tengteng\AppData\
    else x > y:
         ^
SyntaxError: invalid syntax
```

Elif statement

- The *elif* keyword is Python's way of saying "***if the previous conditions were False, then try this condition***".
- Use *elif* statement to specify a new condition to test, if the first condition is false.



Elif statement

- Basic example

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

Variation

- You can use *if…elif…elif* sequence to test many conditions.

```
choice = 8
if choice == 1:
    print('case 1')
elif choice == 2:
    print('case 2')
elif choice == 3:
    print('case 3')
elif choice == 4:
    print('case 4')
else:
    print('default case')
```

```
choice = 2
if choice == 1:
    print('case 1')
elif choice == 2:
    print('case 2')
elif choice == 3:
    print('case 3')
elif choice == 4:
    print('case 4')
else:
    print('default case')
```

Elif statement

- Which will never print?

```
if x < 2:
    print (“Below 2")
elif x >= 2:
    print (“Two or more")
else:
     print (“Something else")
```

Elif statement

- Which will never print?

```
if x < 2:
    print (“Below 2")
elif x >= 2:
    print (“Two or more")
else:
     print (“Something else")
```

```
if x < 2:
    print (“Below 2")
elif x < 20:
    print (“Below 20")
elif x < 10:
    print (“Below 10")
else:
     print (“Something else")
```

Short hand if...else

- If you have only one *if* statement to execute, you can put it all on the same line:

```
x,y = 3,5
if x < y: print('foo')
```

```
foo
```

```
# separate with semicolon
x,y = 3,5
if x < y: print('foo'); print('bar'); print('baz')
```

```
foo
bar
baz
```

Short hand if...else

- If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

```
print('foo') if x > y else print('bar')
```

```
bar
```

- If you have a sequence of conditions to test, you can put each of them on the same lines:

```
if x < y: print('foo')
elif y < x: print('bar')
else: print('baz')
```

```
foo
```

Multiple conditions

- To join two or more conditions into a single if statement, use logical operators -- ***and***, ***or*** and ***not***.
- ***and*** expression is True, if all the conditions are true.

```
# and expression
x, y, z = 7, 4, 2
if x > y and x > z:
    print('x is greater')
```

```
x is greater
```

Multiple conditions

- To join two or more conditions into a single if statement, use logical operators -- ***and***, ***or*** and ***not***.
- ***and*** expression is True, if all the conditions are true.
- ***or*** expression is True, if at least one of the conditions is True.

```
# or expression
x, y, z = 7, 4, 9
if x > y or x > z:
    print('x is greater than y or z')
```

```
x is greater than y or z
```

Multiple conditions

- To join two or more conditions into a single if statement, use logical operators -- ***and***, ***or*** and ***not***.
- ***and*** expression is True, if all the conditions are true.
- ***or*** expression is True, if at least one of the conditions is True.
- ***not*** expression is True, if the condition is false.

```
# not expression
x, y = 7, 5
if not x < y:
    print('x is greater')
```

```
x is greater
```

Lab *Condition*