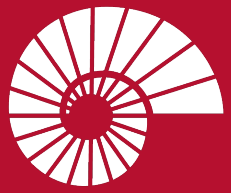


KOÇ
ÜNİVERSİTESİ

04 – Boolean Logic and Decision Structures

COMP125 Programming with Python

Recording Disclaimer

The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

Boolean Expressions

- **Boolean variable** can reference one of the two values, either **True** or **False**
  - Boolean variables are mostly used as flags
    - **True**, if the condition does exist
    - **False**, if the condition does not exist
    - e.g., graduated = False (if you did not graduate yet)
    - e.g., visited = True (if you already visited a particular city)

- **Boolean expression** is an expression whose value is either **True** or **False**
  - Relational and logical operators are used to form Boolean expressions

Relational Operators

Table 3-1 Relational operators

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

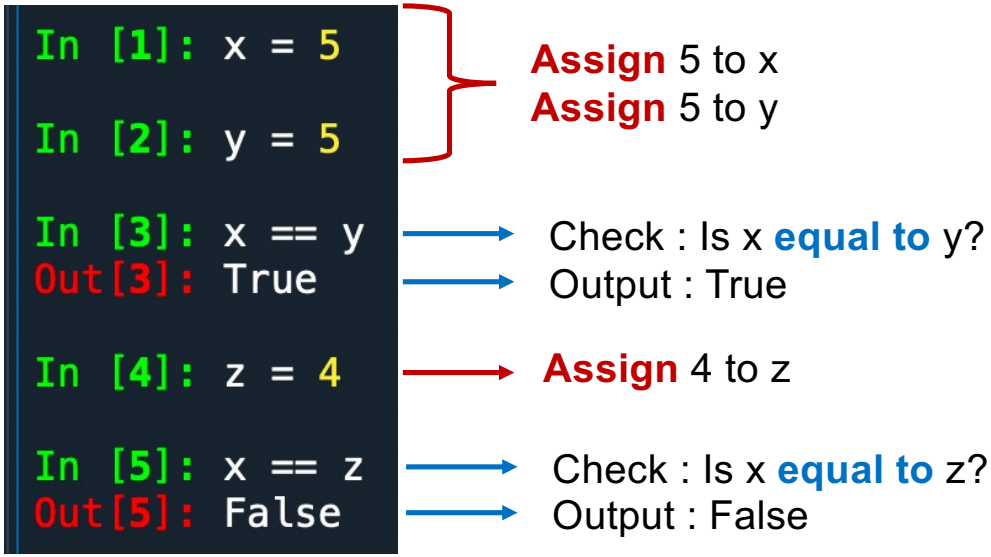
Table 3-2 Boolean expressions using relational operators

Expression	Meaning
x > y	Is x greater than y?
x < y	Is x less than y?
x >= y	Is x greater than or equal to y?
x <= y	Is x less than or equal to y?
x == y	Is x equal to y?
x != y	Is x not equal to y?

Starting out with Python, Tony Gaddis.

Relational Operators

- Do not confuse = with ==
  - = is the **assignment operator**
  - == is the **equality operator** that checks if two of its operands are equal to each other



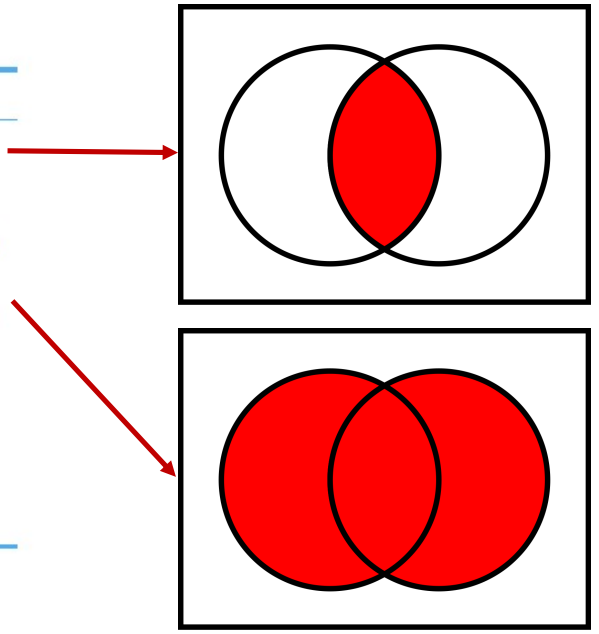
Logical Operators

- **`and`** and **`or`** binary operators connect two Boolean expressions into one compound expression
- **`not`** is a unary operator that negates a Boolean expression

Table 3-3 Logical operators

Operator	Meaning
`and`	The `and` operator connects two Boolean expressions into one compound expression. Both subexpressions must be true for the compound expression to be true.
`or`	The `or` operator connects two Boolean expressions into one compound expression. One or both subexpressions must be true for the compound expression to be true. It is only necessary for one of the subexpressions to be true, and it does not matter which.
`not`	The `not` operator is a unary operator, meaning it works with only one operand. The operand must be a Boolean expression. The `not` operator reverses the truth of its operand. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

Starting out with Python, Tony Gaddis.



Compound Boolean expressions

Table 3-4 Compound Boolean expressions using logical operators

Expression	Meaning
`x > y and a < b`	Is x greater than y AND is a less than b?
`x == y or x == z`	Is x equal to y OR is x equal to z?
`not (x > y)`	Is the expression x > y NOT true?

Starting out with Python, Tony Gaddis.

Truth Tables

X	Y	X or Y	X and Y
False	False	False	False
False	True	True	False
True	False	True	False
True	True	True	True

X	not X
False	True
True	False

Lazy Evaluation

- Python evaluates the binary logical operators as follows
  - It always evaluates the first operand (X)
  - `X and Y`
    - if the value of X is false, it returns the value of X
    - otherwise, it returns the value of Y
  - `X or Y`
    - if the value of X is true, it returns the value of X
    - otherwise, it returns the value of Y

This way of evaluation is sometimes important, e.g., what is the output of the following program when the user enters 0 as the input?

```
N = input('Number: ')
N = int(N)

print( N != 0  and  10 / N )
print( 10 / N  and  N != 0 )
```

Truth Value Testing

- Any object can be tested for its truth value
  - It is the value when it is converted to the Boolean data type using **`bool()`**, *remember the previous class*
- The following have the truth value of **`False`**
  - Constants: **`False, None`**
  - Zero of any numeric type: 0, 0.0, 0j
  - Empty sequences and collections: "", '', [], {}, (), set(), range(0)
- The following have the truth value of **`True`**
  - Constant: **`True`**
  - Numeric values that are not equal to zero
  - Non-empty sequences or collections (lists, tuples, strings, dictionaries, sets)

```
In [1]: bool(0.0)
Out[1]: False

In [2]: bool('')
Out[2]: False

In [3]: bool({})
Out[3]: False

In [4]: bool(range(0))
Out[4]: False

In [5]: bool([])
Out[5]: False

In [6]: bool(-10.4566)
Out[6]: True

In [7]: bool(range(4))
Out[7]: True

In [8]: bool([0, 0 , 0])
Out[8]: True

In [9]: bool('0')
Out[9]: True

In [10]: bool({1, 2, 3})
Out[10]: True
```

Combining Logical Operators

- Python's logical operators have low precedence when compared with other operators
- Precedence of the logical operators from high to low:
  - `not`
  - `and`
  - `or`

```
In [1]: True or True and False
Out[1]: True

In [2]: True or (True and False)
Out[2]: True

In [3]: (True or True) and False
Out[3]: False
```

```
In [4]: "cat" != "dog" or 3 < 4 and 5 == 6
Out[4]: True

In [5]: ("cat" != "dog") or ((3 < 4) and (5 == 6))
Out[5]: True
```

- Consider using parentheses to make the code more readable

Exercise

- What is the output of this code fragment?

```
1   x = 4
2   y = 3
3
4   print( x < y )
5   print( y != 3 )
6   print( x < y or y <= 10 )
7   print( x < y and y <= 10 )
8   print( not(x < y) and y <= 10 )
9   print( x == 5 or x >= y and y != 3 )
10  print( x == 5 or y == 3 and {} or y > 10 )
```

```
False
False
True
False
True
False
False
```

Precedence of the logical operators from high to low

not
and
or

Control Structures

- There are three types of control structures
  - Sequence statements, which are executed sequentially
  - Conditional (decision) statements: **`if, if-else, if-elif-else`**
  - Repetition statements: **`for, while`**
- These statements are combined by either *sequencing* or *nesting*

Sequence Statements

- They are executed sequentially
  - Their execution does not depend on any condition; they are always executed and in the same order

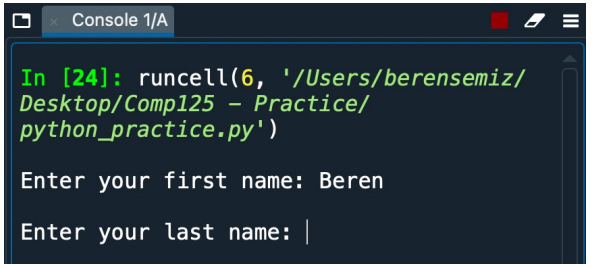
```
1   first_name = input('Enter your first name: ')
2   last_name = input('Enter your last name: ')
3   print('Hello', first_name, last_name, 'welcome to Comp125!')
```

1st line

The following message will appear in the console.

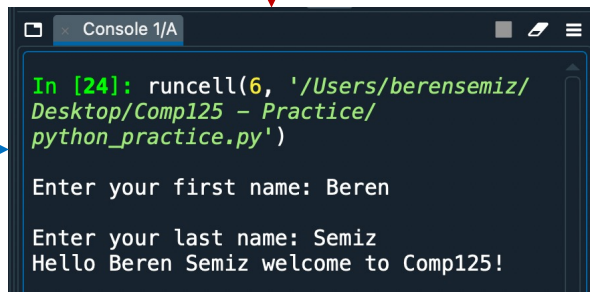
Enter your first name and press ENTER

2nd line



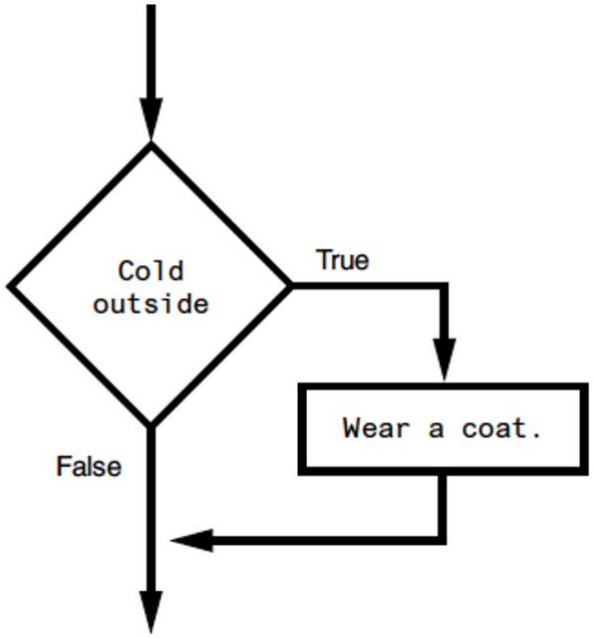
Enter your last name and press ENTER

3rd line



Conditional Statements

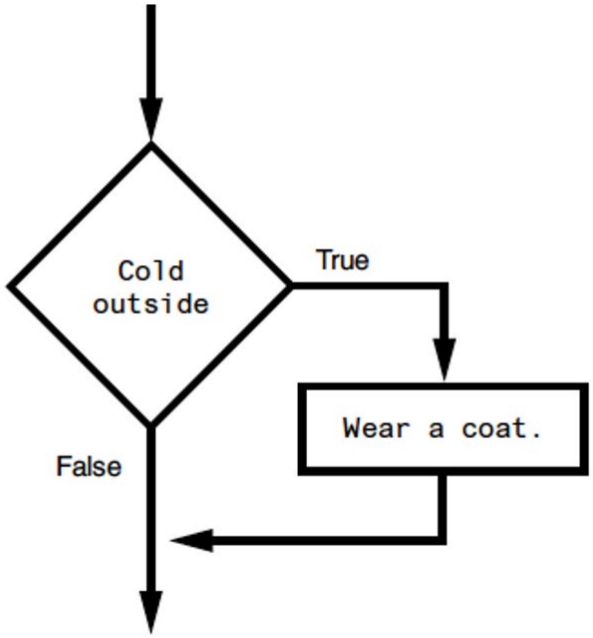
- A specific action is performed only if a certain condition exists (i.e., if a certain condition is `True`)

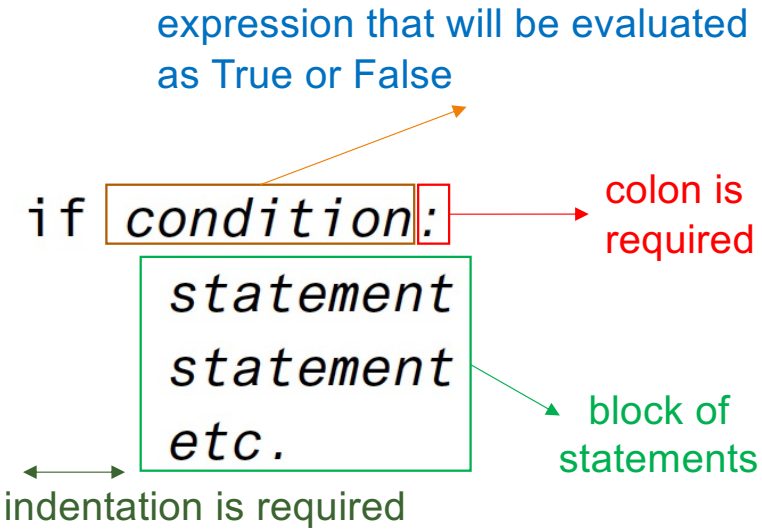


It is said that the action is 'conditionally executed'

`if` statement

- We use the `if` statement to write a **single alternative decision** structure





Example

- Kate teaches a science class and her students are required to take three tests. She wants to write a program that her students can use to calculate their average test score. *She also wants the program to congratulate the student if the average is greater than or equal 80.*

Pseudocode

Get the first test score

Get the second test score

Get the third test score

Calculate their average (their sum divided by 3)

Display the average

If the average is greater than or equal to 80

    Display the congratulation message

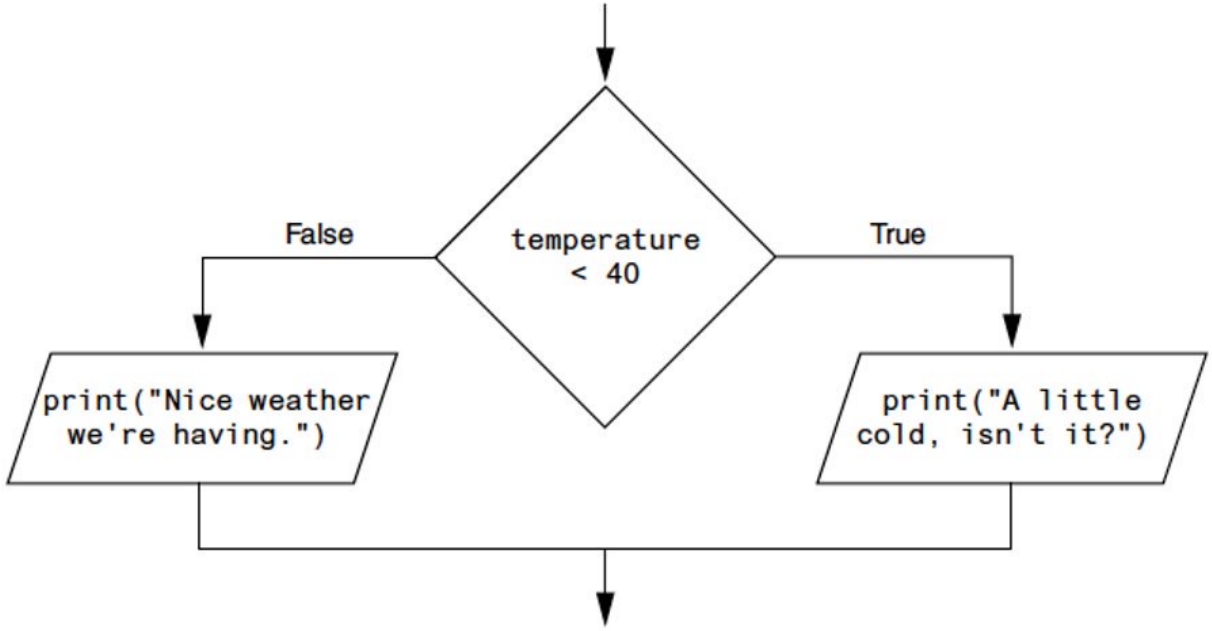
sequential statement (last lecture)

conditional statement (new part)

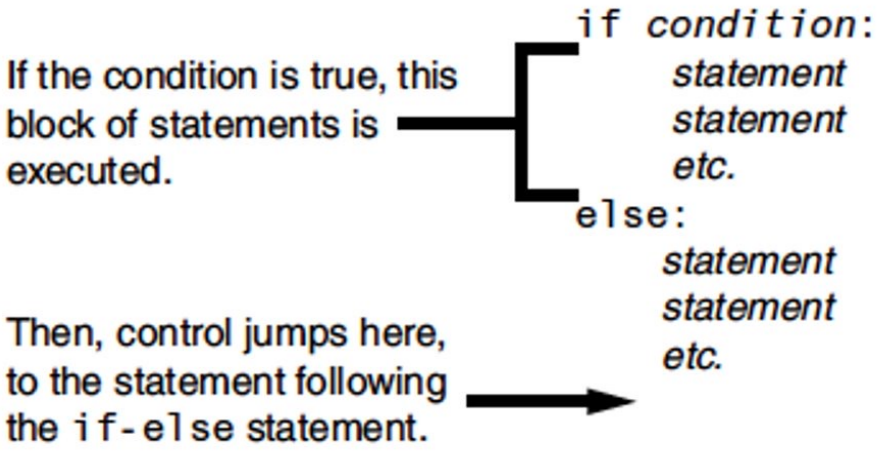
```
first = input('Enter the first test score: ')
second = input('Enter the second test score: ')
third = input('Enter the third test score: ')

first = int(first)
second = int(second)
third = int(third)

avg = (first + second + third) / 3
print('Your test average: ', format(avg, '.1f'))

if avg >= 80:
    print('Congratulations!!!')
```

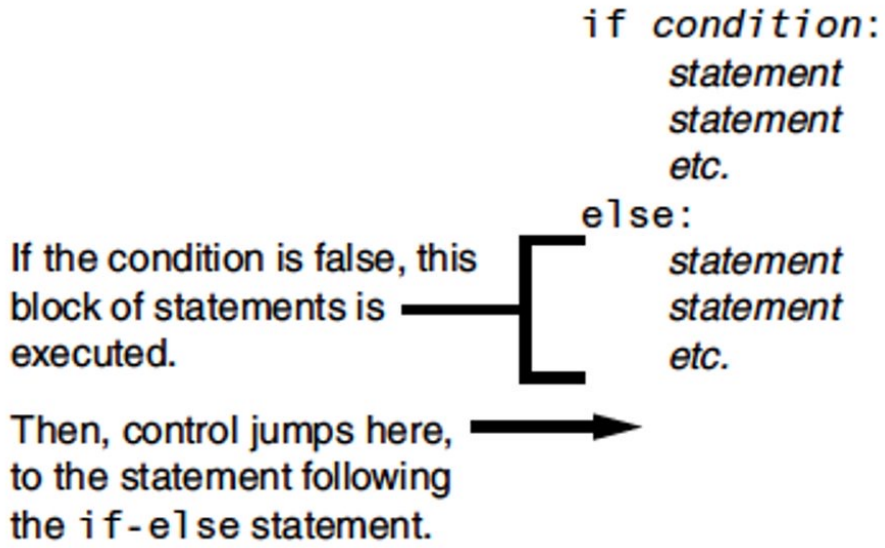
`if-else` statement

- We use the `if-else` statement to write a **dual alternative decision** structure

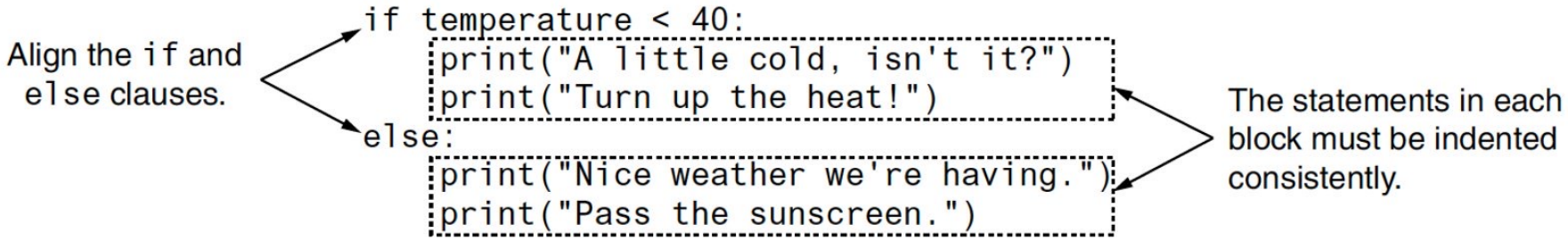


`if-else` statement





`if-else` statement



- **Important rules**:
  - if clause and else clause must be **aligned**
  - The statements following the if clause and else clause must be consistently **indented**

Example

- June 10, 1960, is called a special date because when it is written in the following format, the month times the day equals the year

```
6/10/60

6 * 10 = 60
```

- Design a program that asks the user to enter a month (in numeric form), a day, and a two-digit year, and displays whether it is special. You may assume that all inputs entered by the user are valid.

```
month = input('Month: ')
day = input('Day: ')
year = input('Year: ')

month = int(month)
day = int(day)
year = int(year)

if month * day == year:
    print('Special date :)')
else:
    print('Not special, sorry :(')
```

Example

- Suppose a salesperson has a quota of $50000
- The sales variable references the amount s/he has sold
- Check whether the quota has been met

```
if sales >= 50000.0:
    sales_quota_met = True
else:
    sales_quota_met = False
```

CASE SENSITIVE!!
DO NOT USE
true - false
Capitalize the first letter

```
if sales_quota_met == True:
    print('You have met your sales quota!')
```

SAME

```
if sales_quota_met:
    print('You have met your sales quota!')
```

Overall…

```
if randombool == True:
    statement
```

SAME

```
if randombool:
    statement
```

```
if randombool == False:
    statement
```

SAME

```
if not randombool:
    statement
```

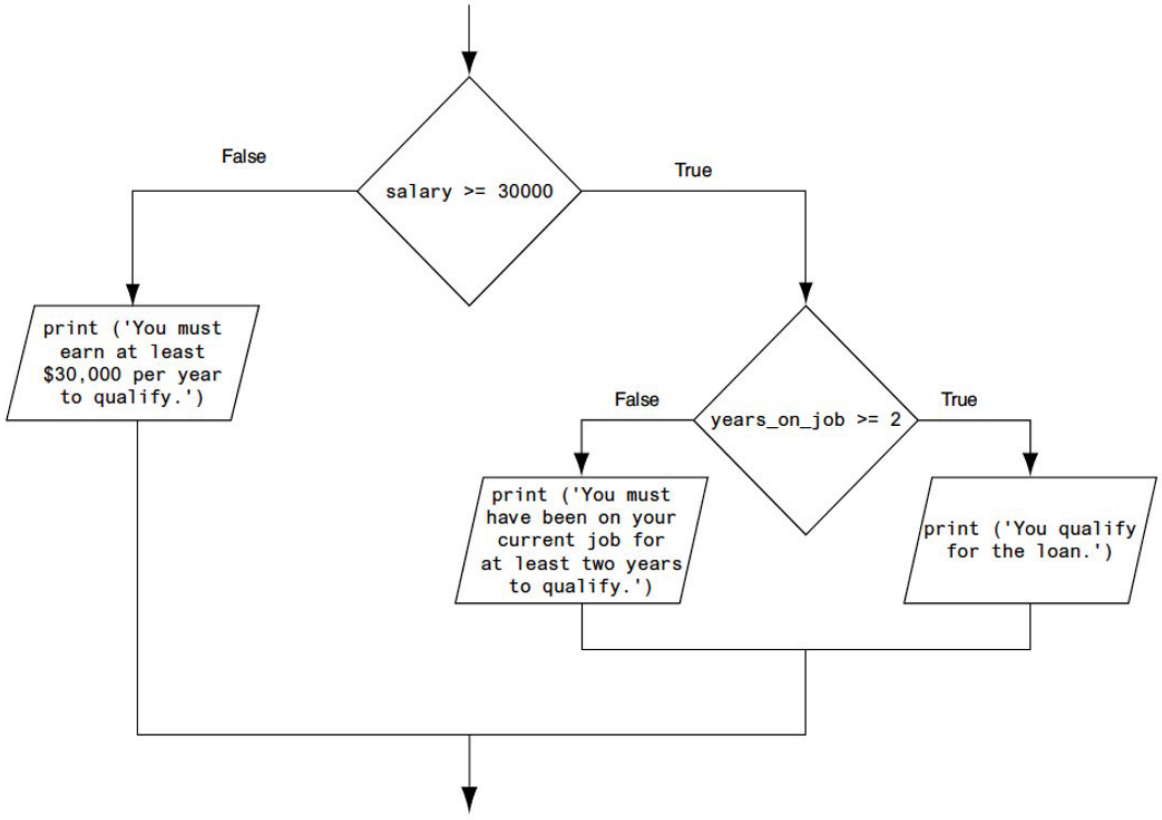
Exercise

- Write a program that reads three floating point numbers and displays their minimum and maximum

```
no1 = input('First no: ')
no2 = input('Second no: ')
no3 = input('Third no: ')

no1 = float(no1)
no2 = float(no2)
no3 = float(no3)

if no1 < no2:
    min_no = no1
    max_no = no2
else:
    min_no = no2
    max_no = no1

if no3 < min_no:
    min_no = no3

if no3 > max_no:
    max_no = no3

print('Min: ', min_no)
print('Max: ', max_no)
```

Nested Conditional Statements

- You write an if (or if-else) statement in the statement block of another if (or else)
- They are used to test more than one condition

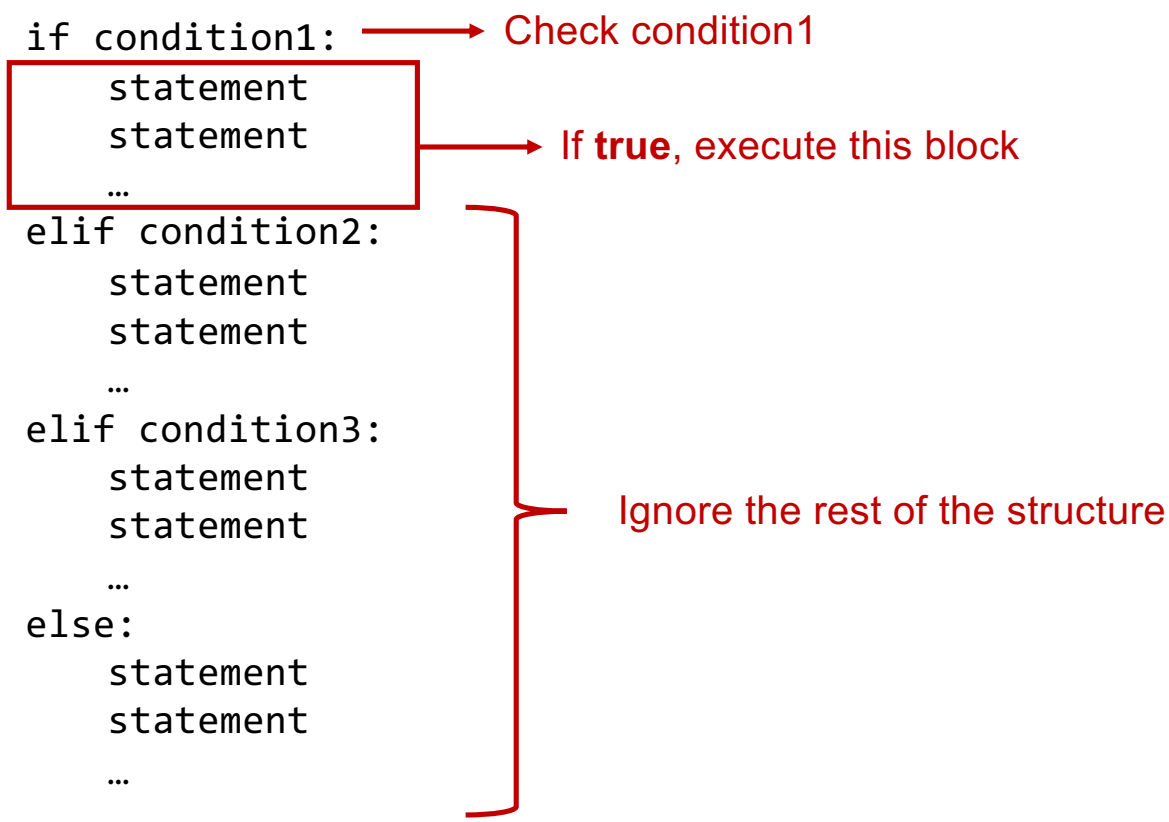


Example: Write a program that determines whether a bank customer qualifies for a loan

```
if salary >= 30000:
    if years_on_job >= 2:
        print('You qualify for the loan')
    else:
        print('You must have been on', end=' ')
        print('your current job for at', end=' ')
        print('least two years to qualify')
else:
    print('You must earn at least', end =' ')
    print('$30,000 per year to qualify')
```

Indentation is used to match the if and else clauses

`if-elif-else` statement



`if-elif-else` statement

```
if condition1:
    statement
    statement
    …
elif condition2:
    statement
    statement
    …
elif condition3:
    statement
    statement
    …
else:
    statement
    statement
    …
```

Check condition1

If **false**, jump to the next `elif` clause and check condition2

If condition2 is **true**, execute this block

Ignore the rest of the structure

Exercise

- Write a program that asks the user to enter an integer, and displays whether it is zero, a positive number, or a negative number

```
num = input('Enter an integer: ')
num = int(num)

if num == 0:
    print('Zero')
elif num < 0:
    print('Negative')
else:
    print('Positive')
```

Exercise

- Write a program that takes a grade, converts it to a letter equivalent according to the following rules, and displays the letter grade on the screen
  - **A : 90 ≤ grade ≤ 100**
  - **B : 80 ≤ grade ≤ 89**
  - **C : 70 ≤ grade ≤ 79**
  - **D : 60 ≤ grade ≤ 69**
  - **F : 0 ≤ grade ≤ 59**
  - **Invalid: All other grades**

```
grade = input('Enter grade: ')
grade = int(grade)

if grade < 0 or grade > 100:
    print('Invalid grade')
else:
    if grade >= 90:
        letter_grade = 'A'
    elif grade >= 80:
        letter_grade = 'B'
    elif grade >= 70:
        letter_grade = 'C'
    elif grade >= 60:
        letter_grade = 'D'
    else:
        letter_grade = 'F'

    print('Letter grade: ', letter_grade)
```

Can you write another version of this algorithm that calculates the letter grade only using the equality operator?

Exercise

- Write a program that takes a grade, converts it to a letter equivalent according to the following rules, and displays the letter grade on the screen
  - `A : 90 ≤ grade ≤ 100`
  - `B : 80 ≤ grade ≤ 89`
  - `C : 70 ≤ grade ≤ 79`
  - `D : 60 ≤ grade ≤ 69`
  - `F : 0 ≤ grade ≤ 59`
  - `Invalid: All other grades`

```
grade = input('Enter grade: ')
grade = int(grade)

if grade < 0 or grade > 100:
    print('Invalid grade')
else:
    grade = grade // 10
    if grade == 10 or grade == 9:
        letter_grade = 'A'
    elif grade == 8:
        letter_grade = 'B'
    elif grade == 7:
        letter_grade = 'C'
    elif grade == 6:
        letter_grade = 'D'
    else:
        letter_grade = 'F'

    print('Letter grade: ', letter_grade)
```

Remember the integer division operator //

Comparing Strings

- Relational operators are also defined on the string data type
  - They compare the Unicode characters of the string from the zeroth index till the end
  - Comparison is case-sensitive

```
name1 = 'Mary'
name2 = 'Mark'

if name1 == name2:
   print('The names are the same')
else:
   print('The names are NOT the same')
```

```
In [1]: 'comp' == 'Comp'
Out[1]: False

In [2]: 'if' >= 'else'
Out[2]: True

In [3]: 'mary' < 'mark'
Out[3]: False

In [4]: 'mark' > 'marking'
Out[4]: False

In [5]: 'january' != 'february'
Out[5]: True
```

Exercise

- Implement a unit converter that takes a value in centimeters and one of the following unit options, converts it to the unit of interest, and displays the equivalent value. You may assume that all inputs are valid.
  - mm, millimeter (1cm = 10mm)
  - cm, centimeter
  - m, meter (1m = 100cm)
  - in, inch (1in = 2.54 cm)

```
value = input('Length (in cm): ')
value = int(value)
option = input('Unit to be converted: ')


if option == 'cm' or option == 'centimeter':
    converted = value
elif option == 'mm' or option == 'millimeter':
    converted = value * 10
elif option == 'm' or option == 'meter':
    converted = value / 10
elif option == 'in' or option == 'inch':
    converted = value / 2.54

print(value, 'cm = ', converted, option)
```