Week 3

Lecture aims:

- Problem Solving
- Use of flowcharts in programming
- Boolean variables and expressions
- Relational operators
- `if`
- `if-else`

Pre-recorded videos:

- `elif` (pre-recorded video)
- Booleans: `and`, `or`, `not` (pre-recorded video)

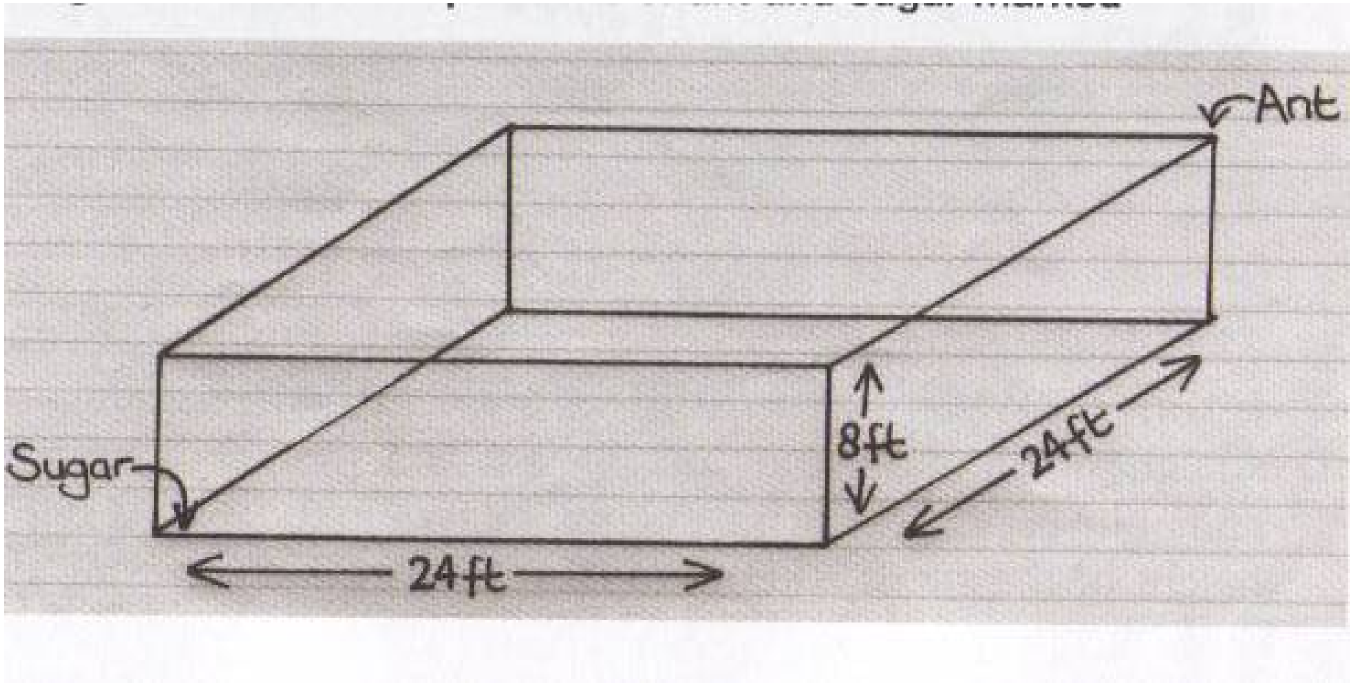
Programming

- Important skill for a computer scientist is **problem solving**:
  - The process of formulating a problem, finding a solution, and expressing the solution.
- How can we solve problems?
  - We need to clearly state the problem
    - Words, diagrams, models, maths ...
  - We need to design the solution before the coding starts

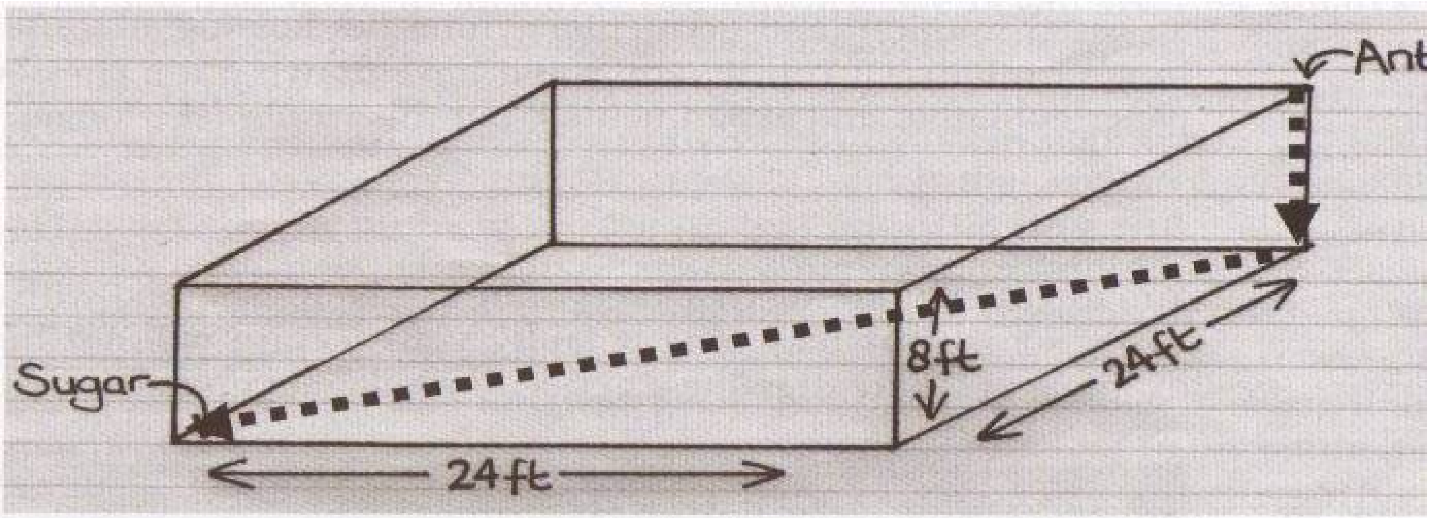
Ant and Sugar Puzzle

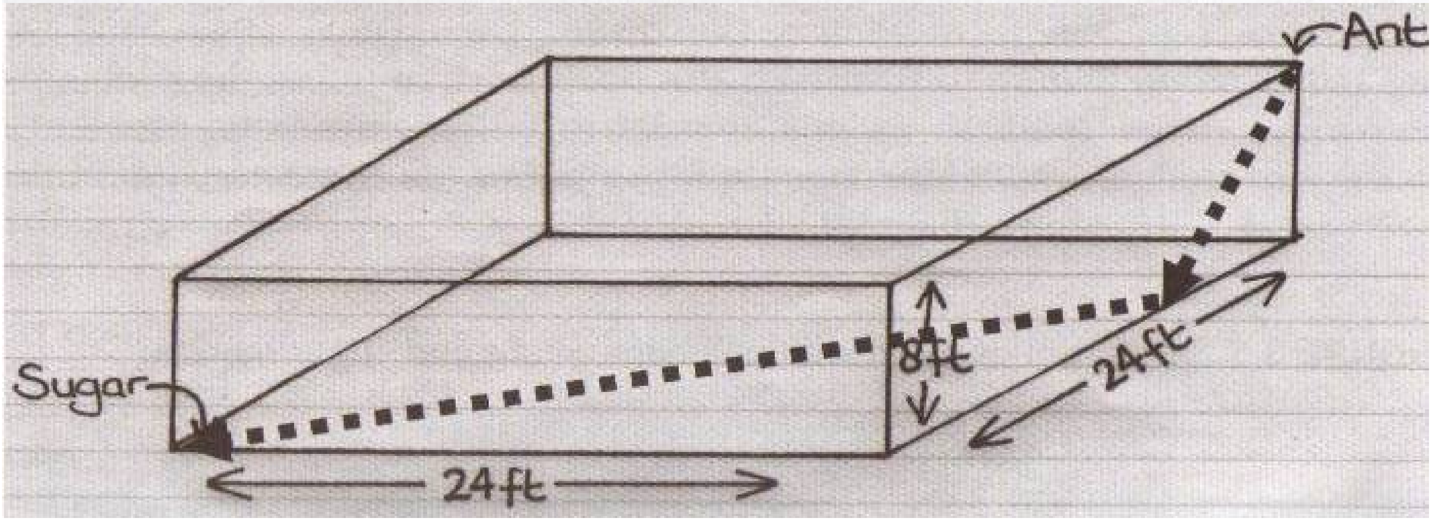
- There is a large square room whose walls are 24 feet long. The ceiling is 8 feet high. On the floor in a corner is a **bowl of sugar**. In the opposite corner by the ceiling is an **ant**.
- What is the shortest path the ant can take to get to the sugar?
  - How shall we start?

3D diagram of room with ant and sugar marked.
…Trace the shortest walking route…

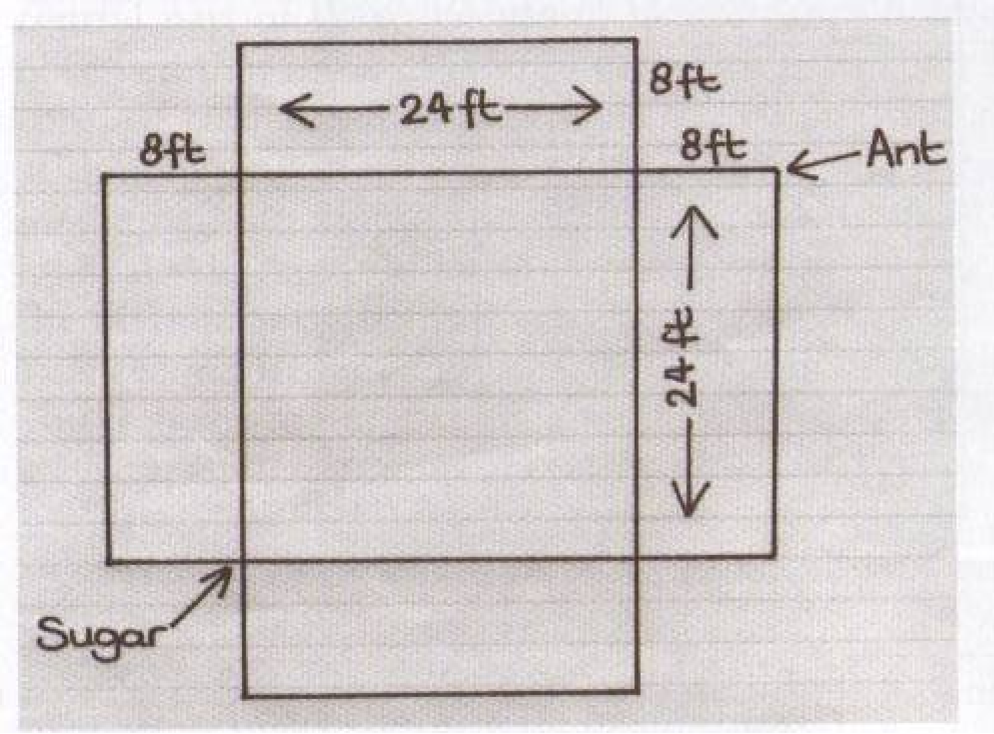


Which is better?

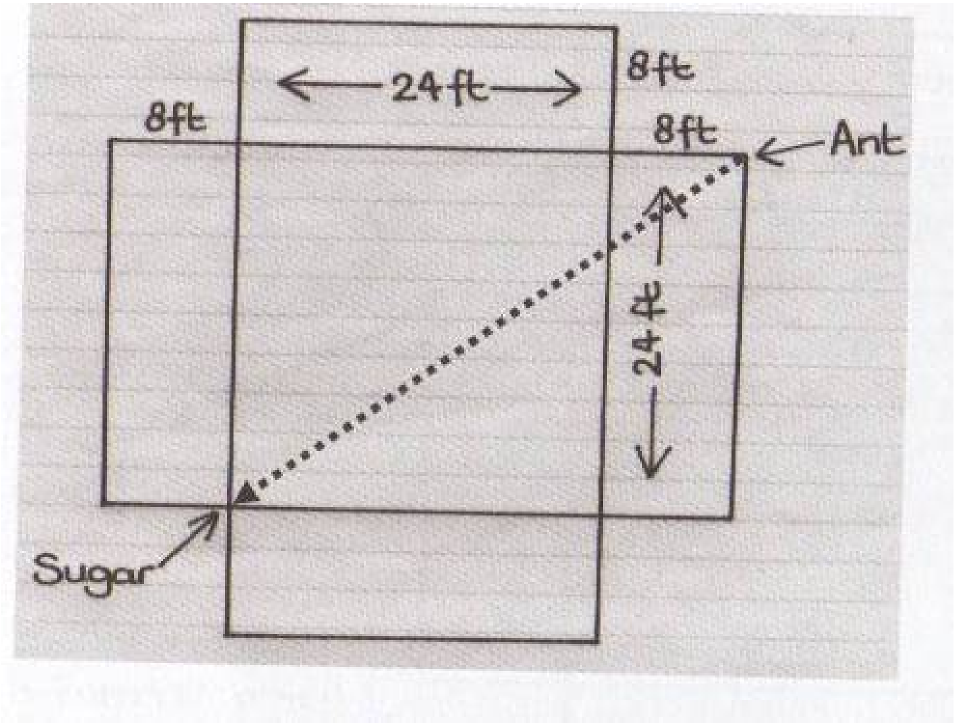




Maybe a different diagram?



Can we be sure we have the best method or solution?



Ant and Sugar Puzzle (Cont’d)

- Made up of two sub problems:
  - Identify shortest path
  - Calculate length of path
    - Maths – Pythagorean theorem

$\text{path}^2 = \text{width}^2 + \text{length}^2$

$\text{path}^2 = 32^2 + 24^2$

$\text{path}^2 = 1024 + 576$

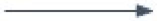
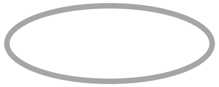
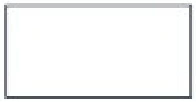
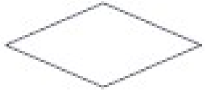
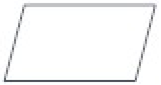
$\text{path}^2 = 1600$

path = 40 ft

Problem Solving: Flowchart Diagrams

- Before you write code, you can use a flowchart to create a diagram of the steps in your algorithm.
- A flowchart shows the structure of decisions and tasks to solve a problem, linked to indicate flow of control.
- Flowchart Diagrams advantages:
  - Decomposition: breaking down a problem into smaller sub problems.
  - Algorithm design: the ability to build a step-by-step process to solve a particular problem.

Common Flowchart elements

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Used to represent start and end of flowchart.
	Process	Used for arithmetic operations and data-manipulations.
	Decision	Used to represent the operation in which there are two alternatives, true and false.
	Input/Output	Used for input and output operation (optional)

Boolean types and expressions

- Python provides the boolean type that can be either set to **True** or **False**. E.g.,

```
finished = True
```

  - We will use the Boolean type during the later loop lecture.

- A boolean expression evaluates to one of two states true or false. E.g.,
- # the operator == tests if two values are equal

```
print(5 == 5)  # produces True
print(5 == 6)  # produces False
```

Conditional Statements

- Sometimes we only want a program to execute code **under certain circumstances.**

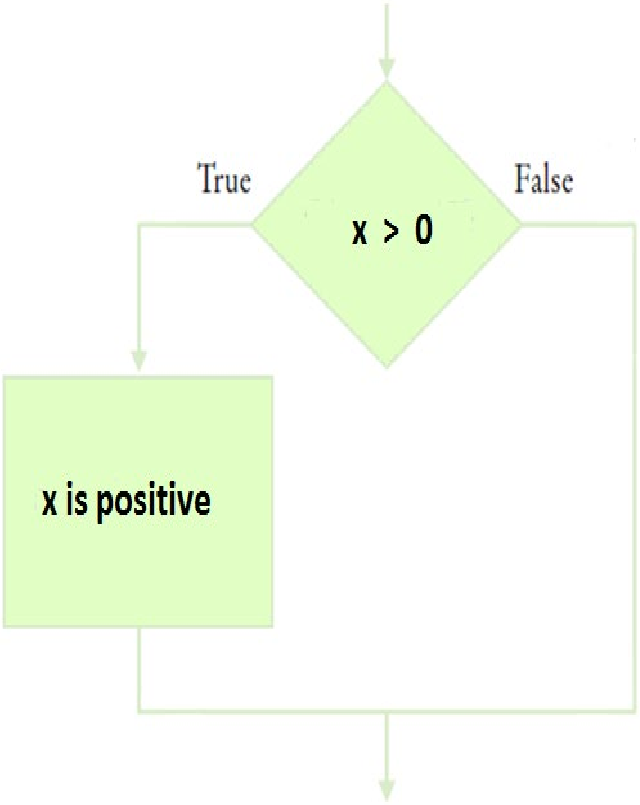
- **Conditional statements** give us the ability to write programs that do different things based on different conditions.

  - `if`
  - `if-else`
  - `elif`

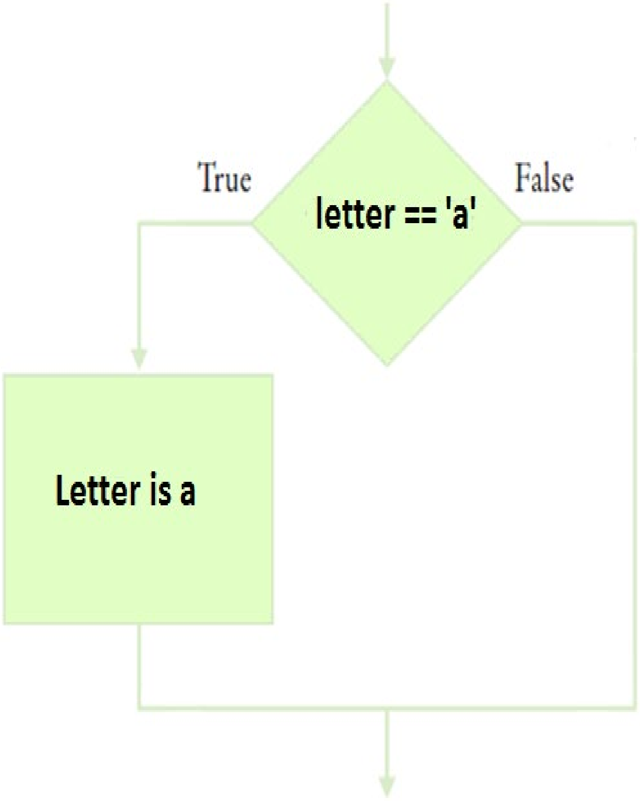
- Note: You may have used a switch case statement in other programming languages. **Python does not have a switch case statement**.

Flowchart - Example 1 & 2

Example 1:



Example 2:



if - Example 1

- **Conditional statements -** simplest form is if:

```
    if x > 0:
        print('x is positive')
```

- **Condition** - the boolean expression after `if`.
  - If it is true, the indented statement runs.
  - If not true, nothing happens.
- **Indent** your print statement so that the program knows that it is part of the if statement. 4-spaces is common choice.

if - Example 2a

- Using one = is setting a variable!
- Using two == is equal to.

```
letter = “a”
if letter == “a”:
    print(“Letter is a”)
```

You must **indent** your print statement so that it is part of the **if statement block**

The **if** must be in **lower case.** You must add a **colon** at the end of the statement.

if - Example 2b

- Now change the program 2a slightly:

```
letter = “b”
if letter == “a”:
    print(“Letter is a”)
```

- Because the answer is **false** it does not print anything

Using indentation

```
letter = “a”

if letter == “a”:
     print(“Letter is a”)
     print(“Prints if letter is a”)
print(“Always prints as not in if block”)
```

block

- The print statement needs to be indented to be applied to the **if statement block.**
- If not indented it is not part of the if block.

Relational operators

- Conditional expressions can be formed using the following operators:

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

Expression is True or False?

Expression	True or False
3 <= 4	
3 =< 4	
3 > 4	
4 < 4	
4 <= 4	
3 == 5 - 2	
3 != 5 - 1	

Exercise 1

- A number a user enters is stored in a variable called `number`.
- Check if `number` is equal to 10.
- If true, display message ‘`number is equal to 10`’.

a) Draw the flowchart first.

b) Then write the program.

Exercise 2 - What is the final value in *b*?

```
a = 3
if a==3 :
    b = a * 2
if a < 4:
    b = a + 2
if a > 2:
    b = a * 2
```

Exercises 3, 4 & 5

3. Which are true if *a* is 3 and *b* is 4?

   a) a + 1 <= b

   b) a + 1 >= b

   c) a + 1 != b

4. Give the opposite of the condition:

```
floor > 13
```

5. What is the error in this statement:

```
if scoreA = scoreB :
    print(“Tie”)
```

Tutorial Q1b

Tutorial Q1b)

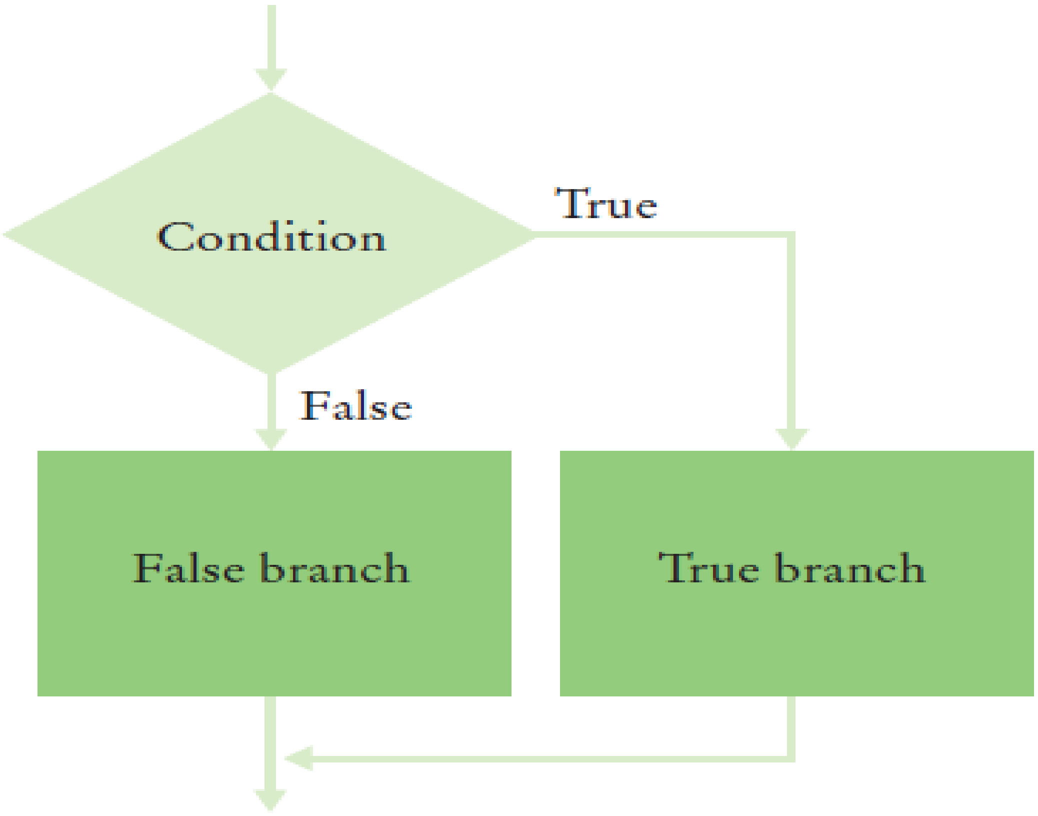
Write a program to read in someone’s age.
Display ‘can vote’ if they are old enough.

a) First create the flowchart (lecture)

a) Write the program (tutorial)

if-else

- Two Outcomes - Get the program to do something if statement is false.
- Flow chart:



if-else: Example 1

- Get the program to do something if statement is false using “else” keyword.

```
a = int(input('Enter number: '))
if a == 10:
    print('a is equal to ten')
else:
    print('a is not equal to ten’)

print('Not in the if or else block')
```

- Print first message if expression is true, otherwise print second message.

if-else: Example 2

- Remember **ZeroDivisionError?**
  - – Error if attempt to **divide** by zero.
- Here is an example solution.

```
b = int(input("Number for division: "))
if b != 0 :
    x = 100/b
else:
    print("Could not divide by zero")
```

if-else: Example 2 - alternative

- **ZeroDivisionError?**
  – Error if attempt to **divide** by zero.
- Alternative solution with try and except.

```
b = int(input("Number for division: "))
try:
    x = 100/b
    print('Answer is', x)
except ZeroDivisionError:
    print("Could not divide by zero")
```

Tutorial Q2b

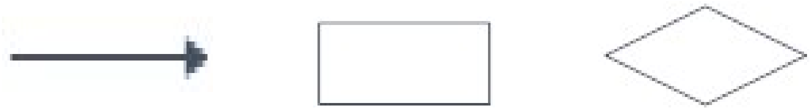
- Tutorial Q2b. Write a program to display “FAIL” if the mark entered is less than 40, otherwise it should display “PASS”.

  a) Create the flowchart (lecture).

  a) Then write the code (tutorial).

Lecture covered

- Problem Solving
- Use of flowcharts in programming:



- Boolean variables and expressions
- Relational operators:

  <   <=   >   >=   ==   !=

- The difference between *if*, *if-else*

Pre-recorded videos:

- `elif` (pre-recorded video)
- Booleans: `and`, `or`, `not` (pre-recorded video)

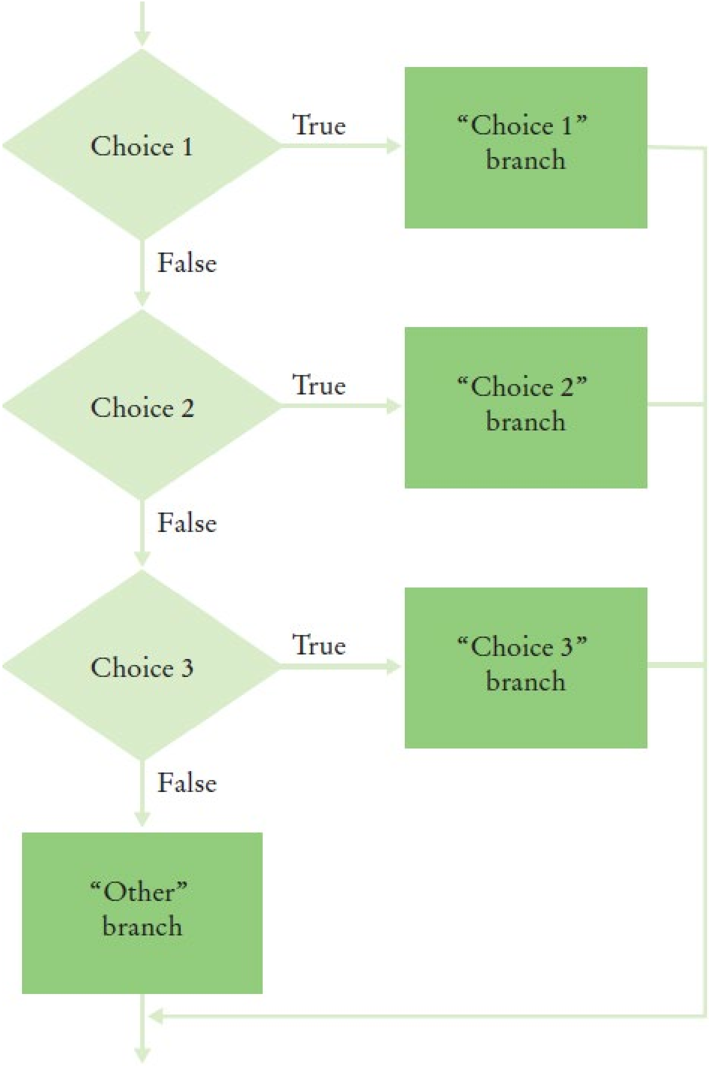
elif

- Pre-recorded video

- adding additional conditions with *elif*.

elif – Additional Conditions

- Give the program additional conditions.
- See general flowchart on the right.

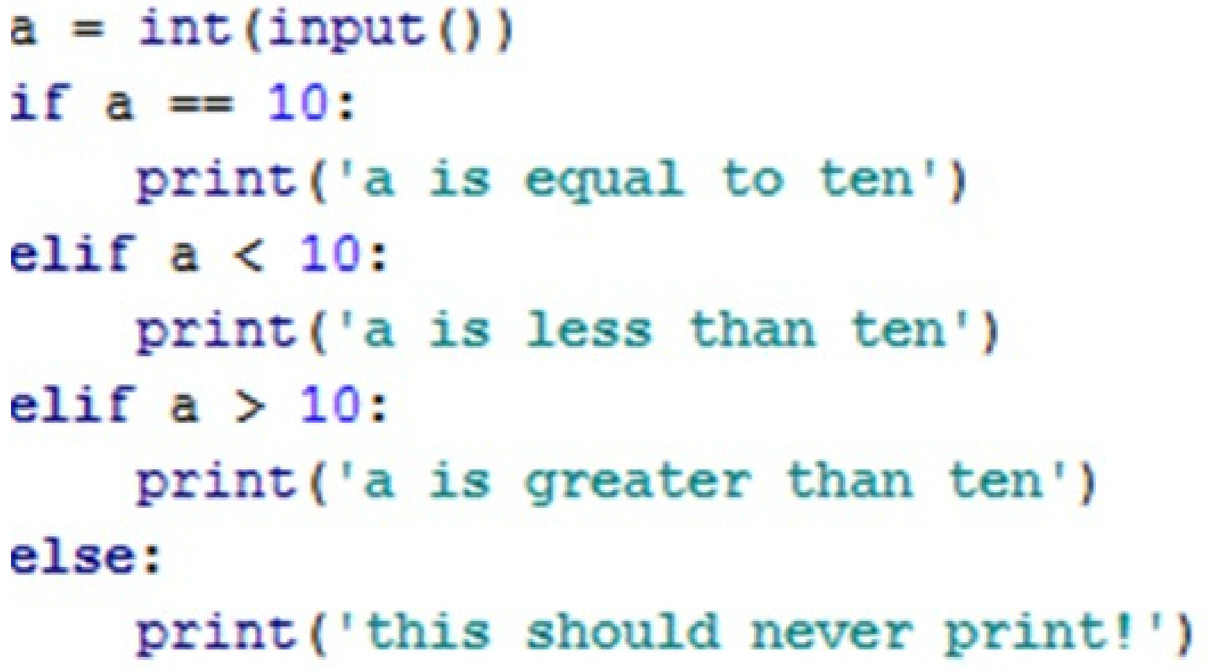


elif (Example 1)

- elif allows you to give the program another condition to try.
- If none match then run else-block (optional)

```
letter=“c”
if letter == “a”:
     print (“Letter is a”)
elif letter == “b”:
     print(“Letter is b”)
else:
     print(“Letter is not a or b”)
```

elif (Example 2)



```
a = int(input())
if a == 10:
    print('a is equal to ten')
elif a < 10:
    print('a is less than ten')
elif a > 10:
    print('a is greater than ten')
else:
    print('this should never print!')
```

- Can use multiple `elif`’s.
- Test conditions for the first match. Only the **first** true branch runs (even if other conditions true).
- If none match then run else-block (optional).

Self-Check Question

```
mark = int(input('Enter Mark'))
if mark >= 40:
    print('Satisfactory result')
else:
    print('You have failed')
```

- Amend the above to display a message for high marks (70 or above)….. Two options on next slide.

Which solution?

```
#1   if mark >= 70:
         print(’Exceptional result’)
     if mark >= 40:
         print(’Satisfactory result’)
     else:
         print(’You have failed’)

#2   if mark >= 70:
         print(’Exceptional result’)
     elif mark >= 40:
         print(’Satisfactory result’)
     else:
         print(’You have failed’)
```

Self-Check Question

- Add an additional condition at the start of the program to check if the exam mark is invalid (less than 0 or greater than 100):

```
if mark >= 70:
    print(’Exceptional result!’)
elif mark >= 40:
    print(’Satisfactory result!’)
else:
    print(’You have failed.’)
```

Testing

- **Testing your programs**
  - Include a test of the boundary of your program decisions.
  - E.g., if a decision checks whether an input is less than 100, test with an input of 99 and 100.

Booleans operators: `and`, `or`, `not`

- `pre-recorded video`

Boolean: `and`

- Evaluate two expressions. Evaluates to True if *both* of the two values are True.

```
x = 10
y = 20
print(x == 10 and y == 20)   #True
```

- *Truth table:*

A and B	Evaluates to
True and True	True
True and False	False
False and True	False
False and False	False

Boolean: and example

- This program validates user input

```
x=int(input(“Enter a number between 1
  and 100:”))
if x>=1 and x<=100:
     print(“Valid number”)
else:
     print(“Your number is not valid”)
```

- Trace which lines would be printed for x: **0, 1, 100, 101**

Boolean: `or`

- Evaluate two expressions. Evaluates to True if *either* of the two values is True.

```
x = 10
y = 20
print(x == 3 or y == 20)     #True
```

- *Truth table:*

A or B	Evaluates to
True or True	True
True or False	True
False or True	True
False or False	False

Boolean: `not`

- not operator evaluates to the opposite Boolean value
  - Inverts a condition.
    - True expressions evaluate to False
    - False expressions evaluate to True.

```
x = 10
print(not x == 10)      #False
print(not x == 3)       #True
```

- *Truth table:*

not A	Evaluates to
not True	False
not False	True

Boolean Variable/ Operators:

- This tests whether both *x* and *y* are zero:

```
    x == 0 and y == 0
```

1. Now state how you test whether at least one of *x* and *y* is zero.

2. Which is correct - a, b or c?

```
a) age < 17 or > 150            #don’t drive
b) age < 17 or  age > 150       #don’t drive
c) age < 17 and age > 150       #don’t drive
```