3. FLOW CONTROL - IF

VICTOR LEE

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

DECISION AND ACTION

- In real-life:
  - We make decisions almost everyday
  - Decisions will be followed by one or more actions
- In programming:
  - Decision is based on a condition (logical expression) that is either true or false
  - Action is in form of program statements

WHAT MAKE A DECISION IN PYTHON

- Understand the problem
  - Understand the problem such as requirements and constraints
- Identify the possible alternatives
  - Develop an algorithm to solve the problem (make a decision)
- Formulate conditions for the alternatives
  - Based on a condition (logical expression) which is either true or false to select an alternative
- Take action!
  - Write code for each alternative

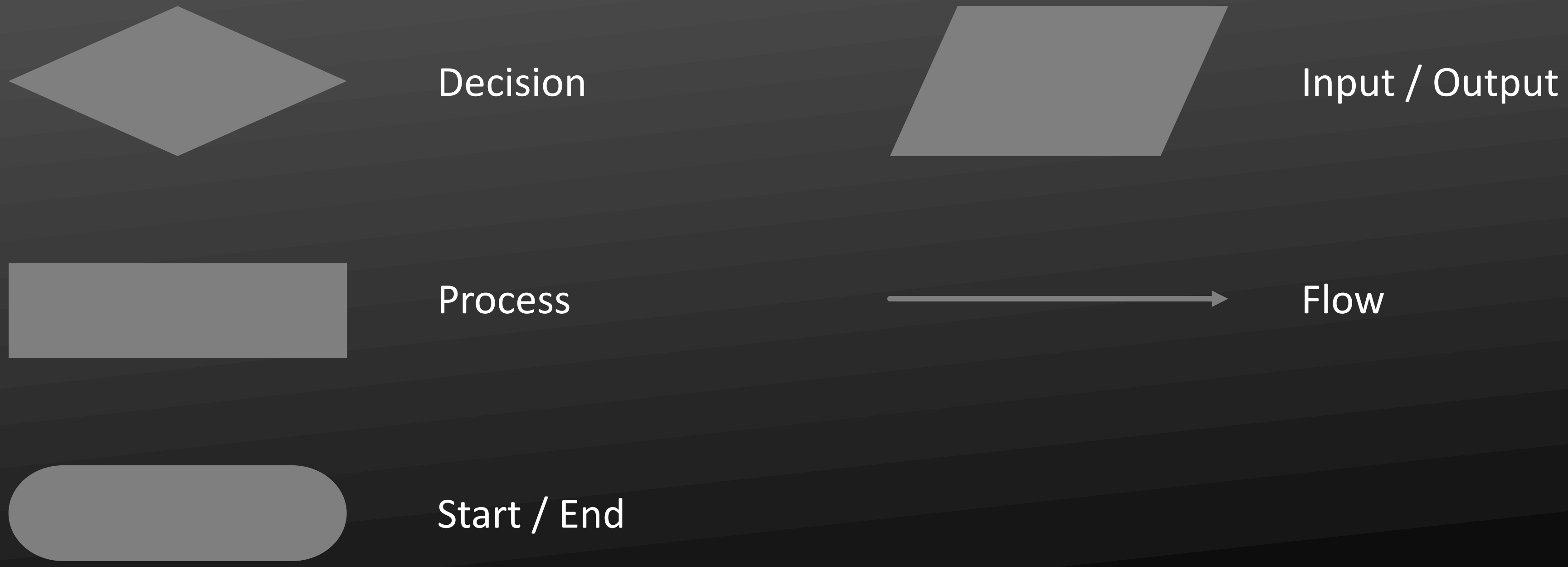
OUTLINES

- Flow chart
- If statement
    - Simple
    - Chained
    - Nested
- Logical expression

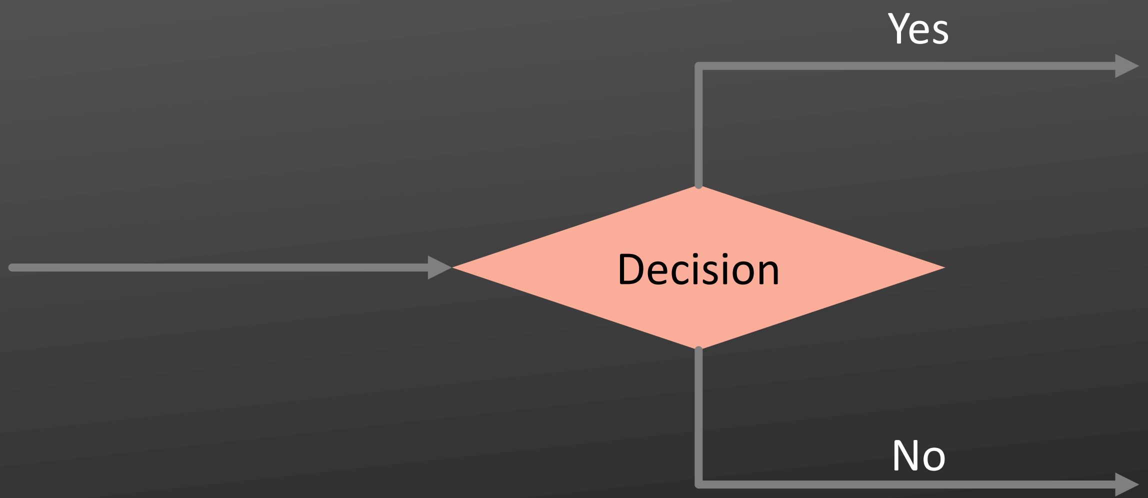
HOW TO DEPICT YOUR THOUGHT(LOGIC)?

FLOWCHART

- A tool used to visualize the logic flow of an algorithm/process
- Basic elements:

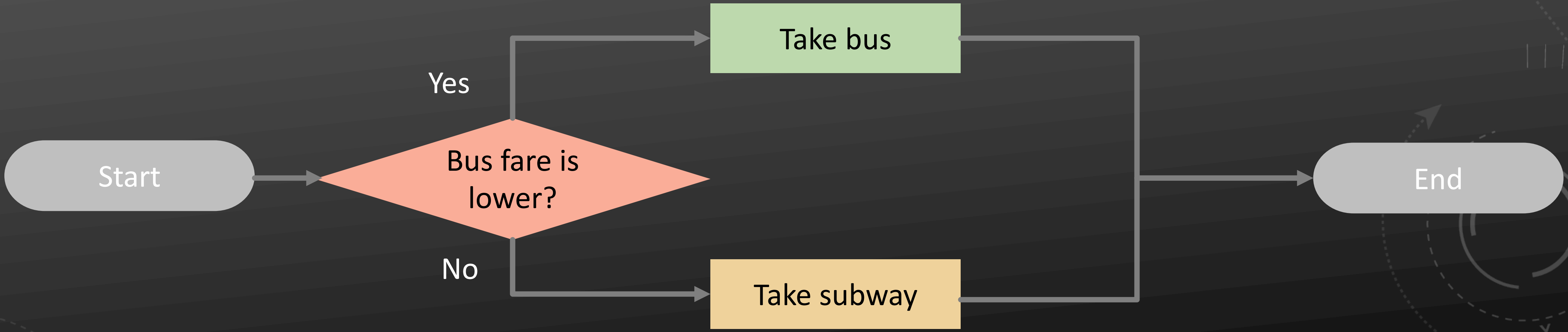


THE BASIC: TO MAKE A DECISION BETWEEN TWO ALTERNATIVES



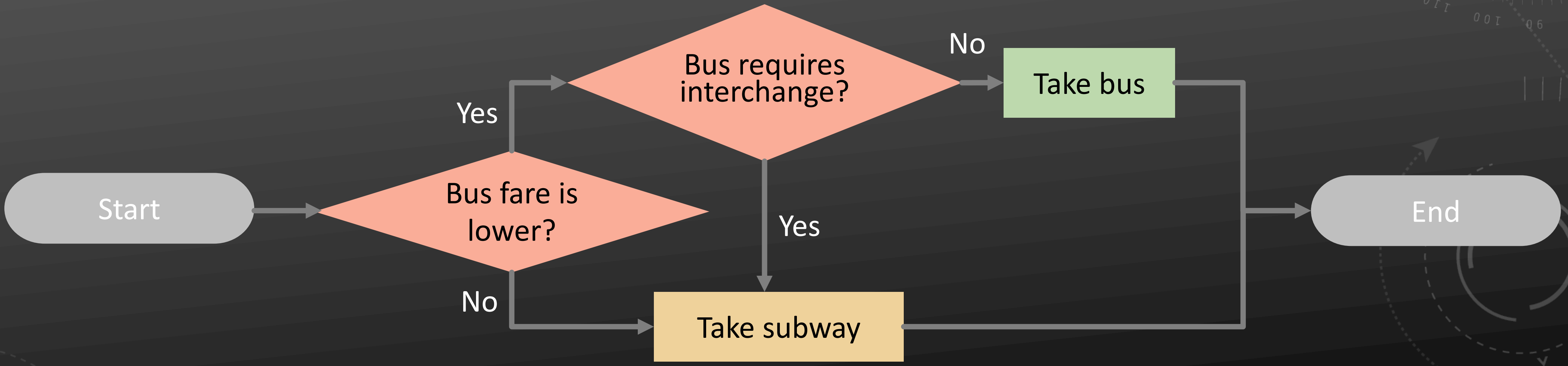
EXAMPLE

- Make a decision between bus and subway based on cost

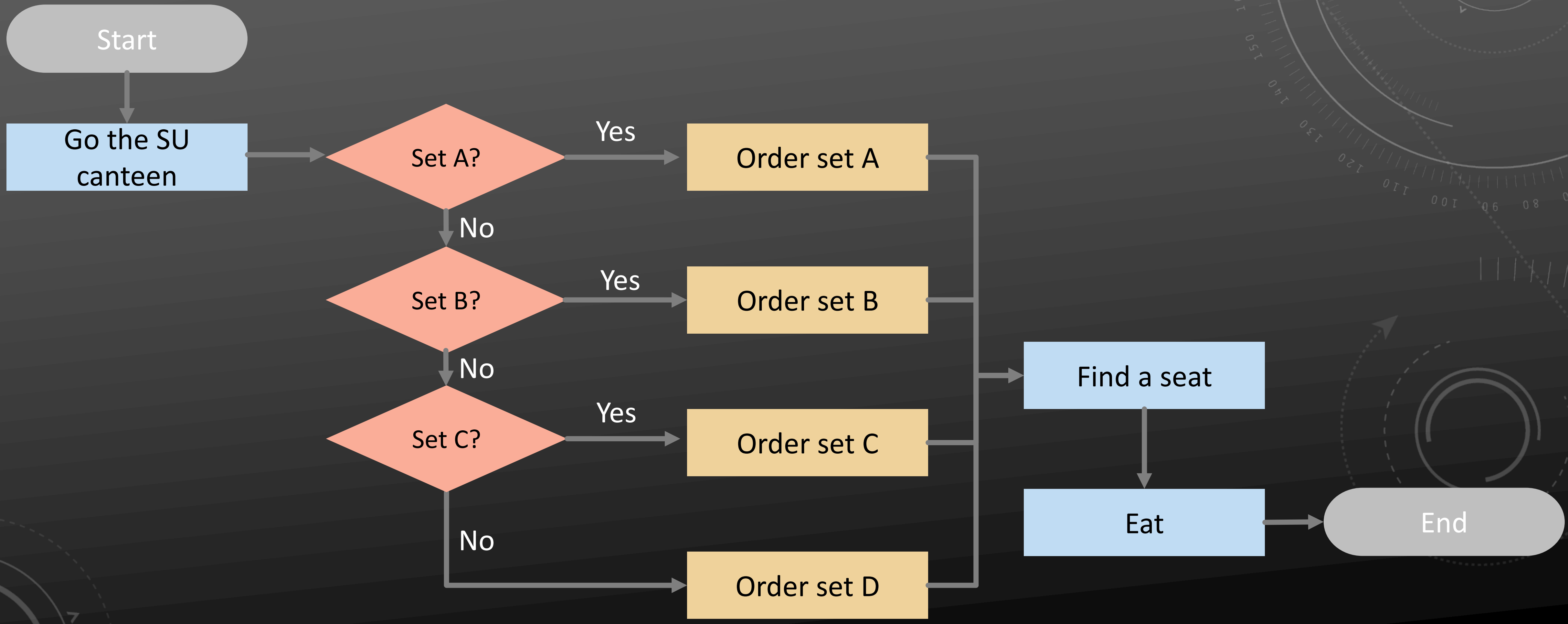


EXAMPLE

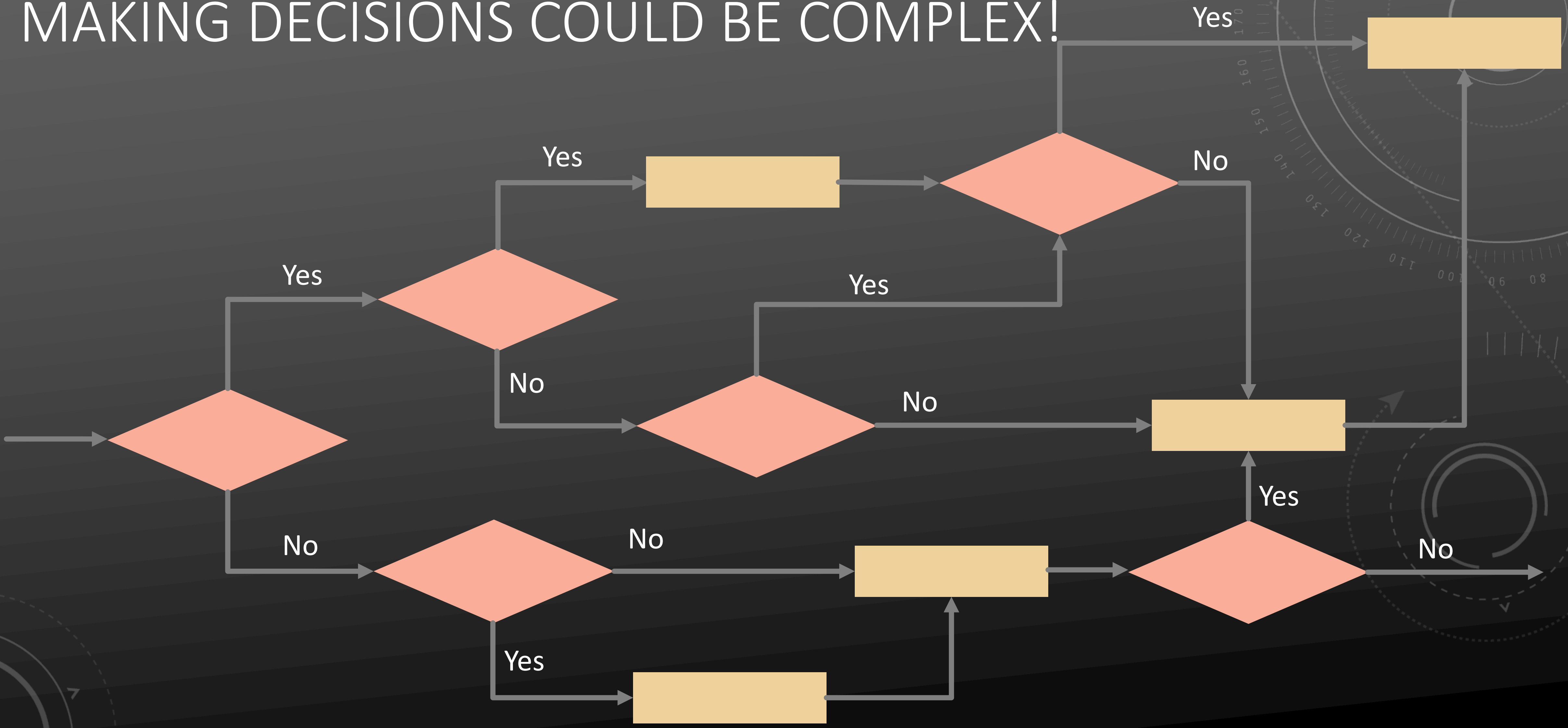
- Make a decision between bus and subway based on cost and need of interchange



DID YOU HAVE TROUBLE TO MAKE DECISIONS?



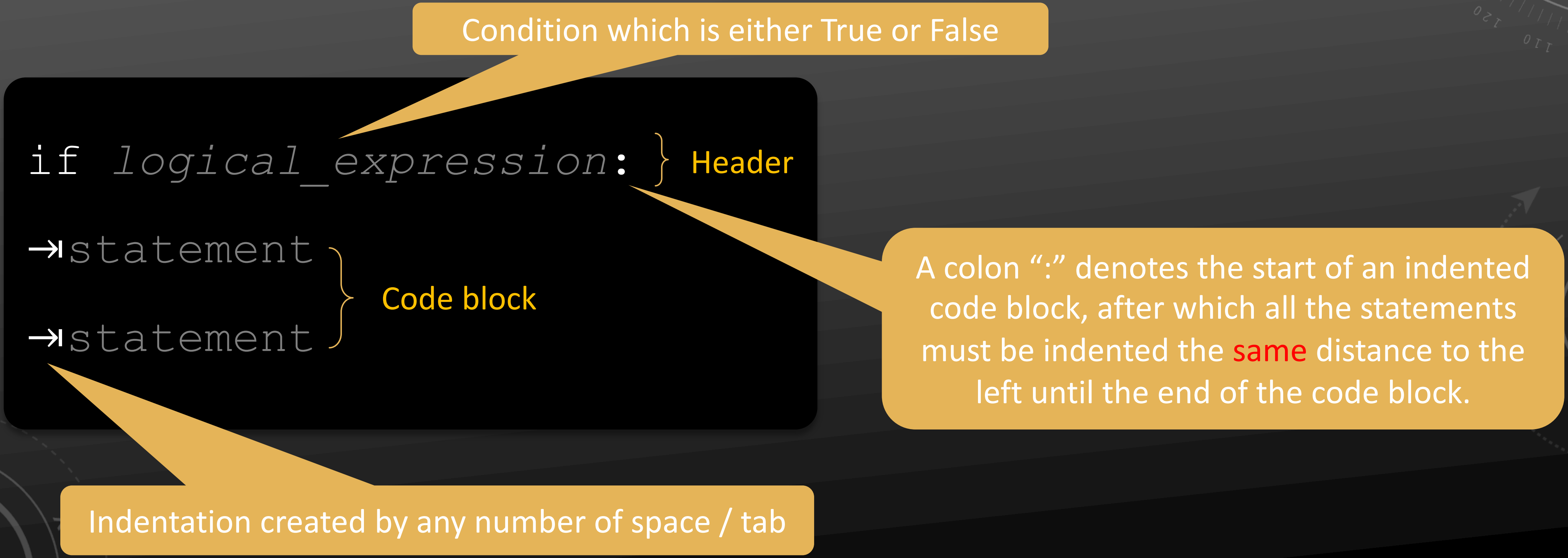
MAKING DECISIONS COULD BE COMPLEX!



MAKING DECISION IN PYTHON

SIMPLE IF STATEMENT

- If the condition (logical expression) is true, the **indented code block** runs.
- If not, nothing happens.



Condition which is either True or False

```
if logical_expression:
→statement
→statement
```

A colon ":" denotes the start of an indented code block, after which all the statements must be indented the same distance to the left until the end of the code block.

Indentation created by any number of space / tab

EXAMPLE: PRINT THE ABSOLUTE VALUE OF AN INTEGER

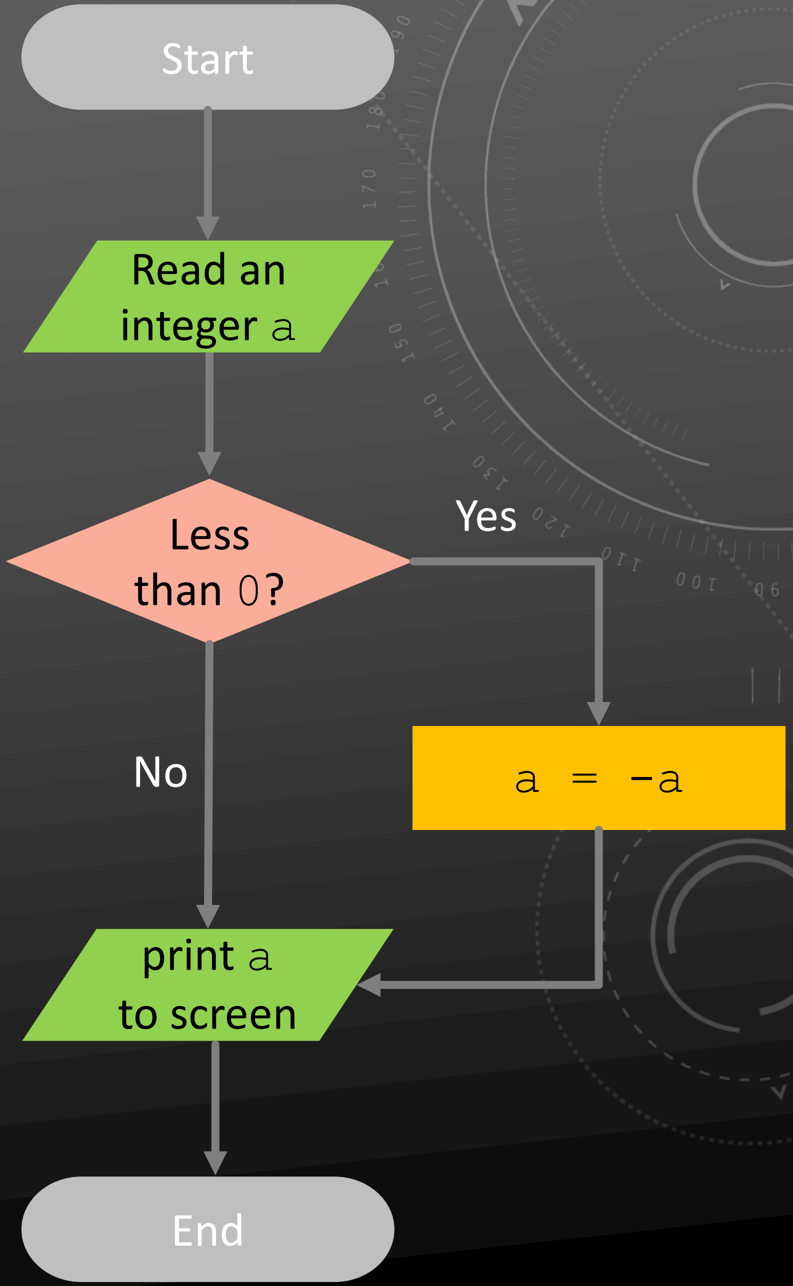
```
a=int(input())

if a<0:
    a=-a

print(a)
```

```
4
4
```

```
-3
3
```



SIMPLE IF-ELSE STATEMENT

- If the condition is true, the **first** code block runs.
- If not, the **second** code block runs.
- The two alternatives are called **branches** because they are branches in the flow of execution.

```
if logical_expression:

⇥statement  }
⇥statement  }  Code block to be executed if the condition is true

else:

⇥statement  }
⇥statement  }  Code block to be executed if the condition is false
```

EXAMPLE: EVEN NUMBER?

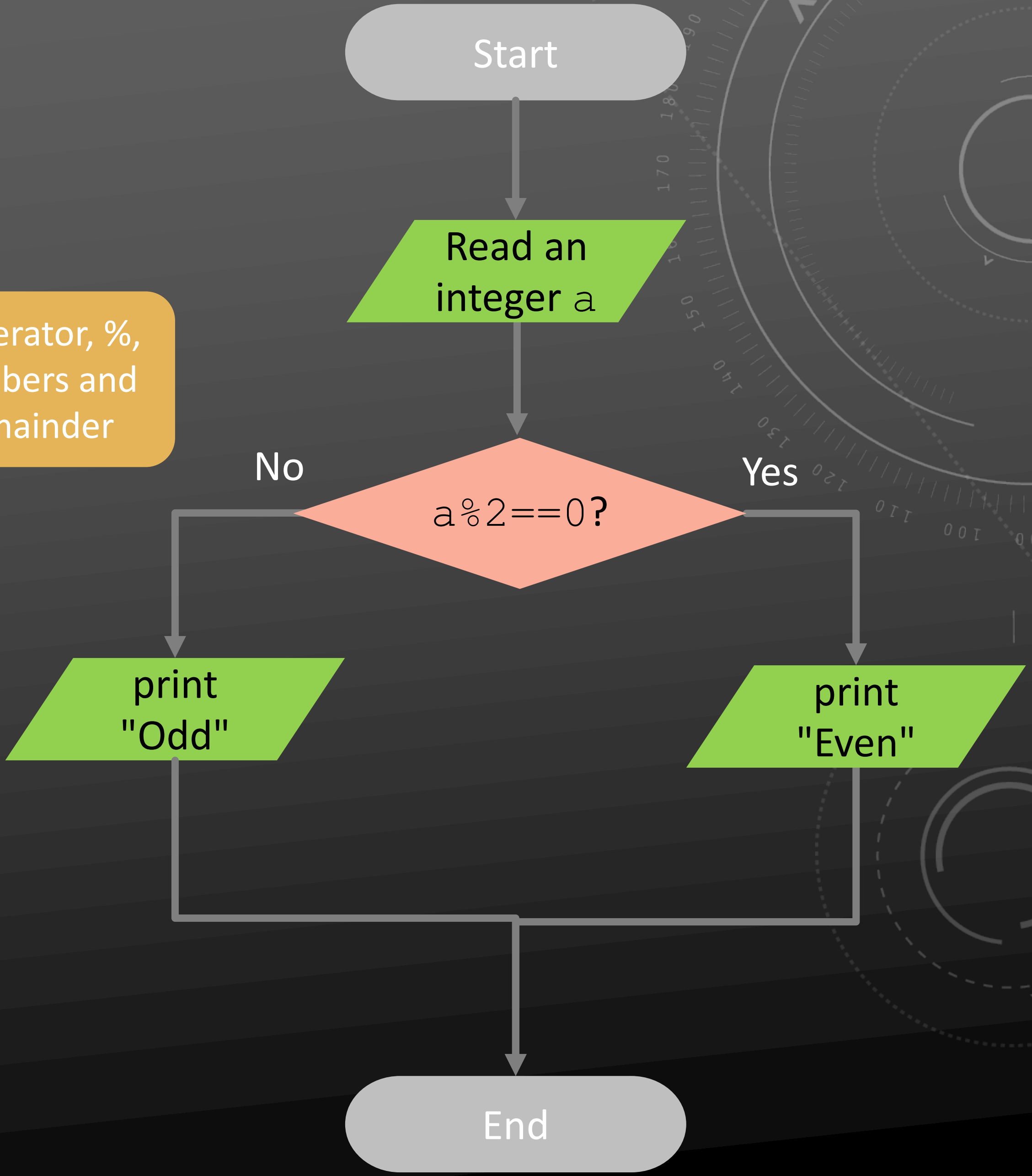
```
a=int(input())

if a%2==0:
   print("Even")
else:
   print("Odd")
```

The modulus operator, %, divides two numbers and returns the remainder

```
4
Even
```

```
-3
Odd
```



CHAINED CONDITIONALS

```
if logical_expression_1:

→statement
→statement

elif logical_expression_2 :

→statement
→statement

else:

→statement
→statement
```

Code block to be executed if the logical expression 1 is false but logical expression 2 is true

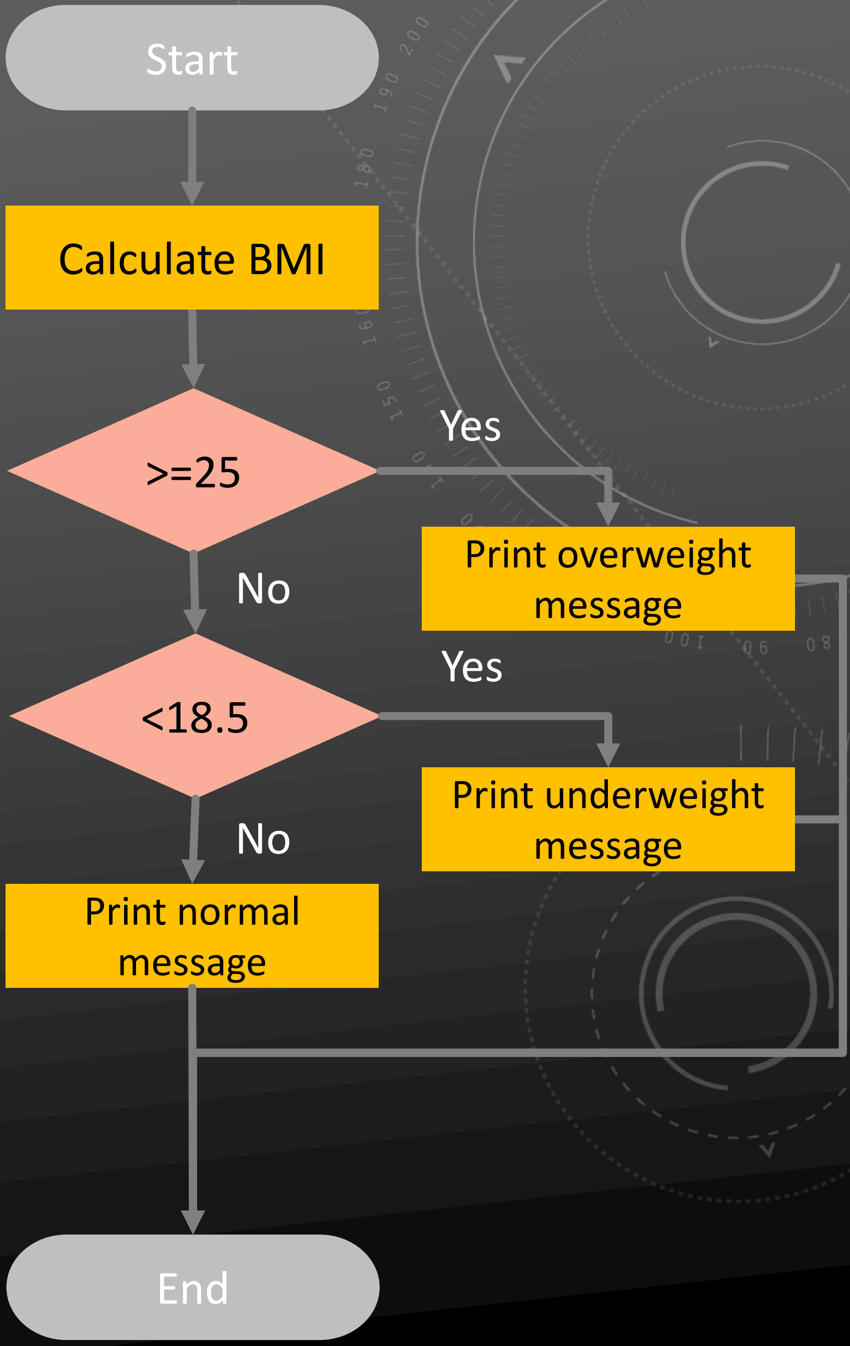
If there is an `else` clause, it has to be at the end

There is no limit on the number of `elif` but only the code block of the first true condition runs.

EXAMPLE: AM I FAT?

```
weight=float(input())
height=float(input())
bmi=weight/height/height

if bmi>=25:
    print("Overweight")
    print("You should do more exercise")
elif bmi<18.5:
    print("Underweight")
    print("Please eat more")
else:
    print("Normal")
    print("Good!")
```



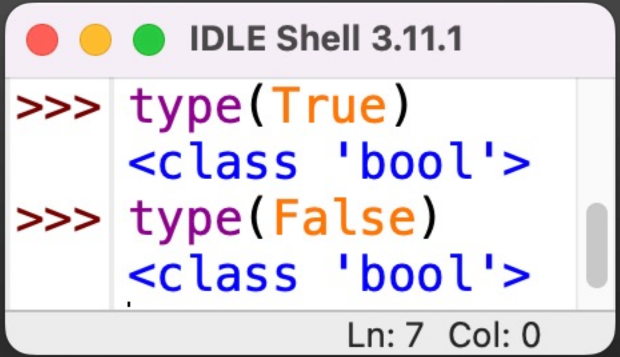
NESTED CONDITIONALS

- One conditional appears in one of the branches of another conditional
- Statements indented the same distance to the left belong to the same code block

```
if bmi>=25:
    print("Overweight")
    print("You should do more exercise")
else:
    if bmi<18.5:
        print("Underweight")
        print("Please eat more")
    else:
        print("Normal")
        print("Good!")
```

LOGICAL EXPRESSION

- A logical expression is either true or false
- `True` and `False` are special values of the Boolean type `bool`
- `True`
  - Nonzero number
  - Nonempty object
- `False`
  - A zero number
  - Empty object
  - None
- Comparative operators (>, <, >=, =<) and logical operators (`and`, `or`) return a `True` or `False`



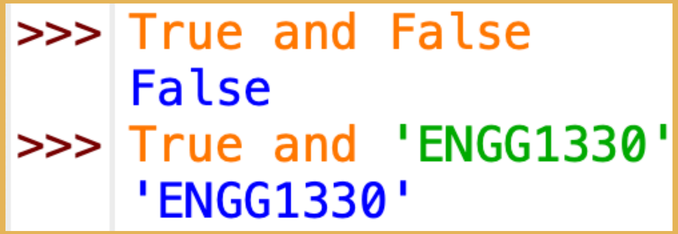
```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

LOGICAL OPERATORS

- `and`
  - Return `True` if **both** operands are true
  - `x and y`
- `or`
  - Return `True` if **either** operand is true
  - `x or y`
- `not`
  - Return `True` if the operand is false
  - `not x`

Operand is the value on which an operator operates

Logical operators return the last evaluated operand if it is not a Boolean value.



```
>>> True and False
False
>>> True and 'ENGG1330'
'ENGG1330'
```

LOGICAL OPERATORS

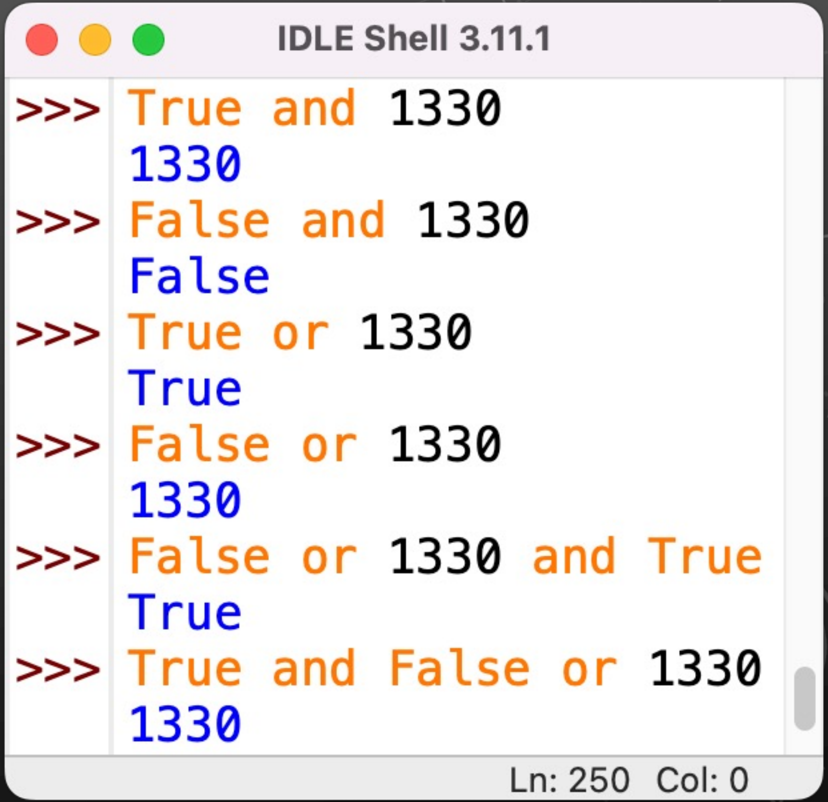
x	y	x and y
True	True	True
True	False	False
False	True	False
False	False	False

x	y	x or y
True	True	True
True	False	True
False	True	True
False	False	False

x	not x
True	False
False	True

SHORT-CIRCUIT EVALUATION

- ❑ Evaluation of expressions containing `'and'` and `'or'` stops as soon as the outcome `True` or `False` is known and this is called *short-circuit evaluation*
- ❑ Short-circuit evaluation can improve program efficiency
- ❑ Short-circuit evaluation exists in some other programming languages too, e.g., C++ and Java



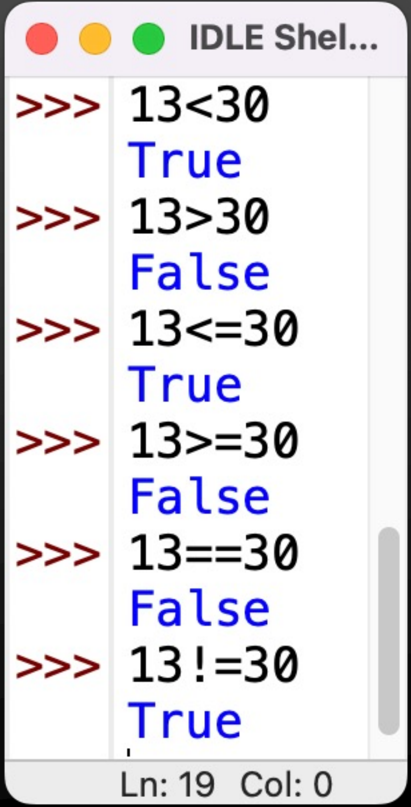
```
>>> True and 1330
1330
>>> False and 1330
False
>>> True or 1330
True
>>> False or 1330
1330
>>> False or 1330 and True
True
>>> True and False or 1330
1330
```

COMPARATIVE OPERATORS

- Binary operators which accept two operands and compare them, return either `True` or `False`

Relational operators	Syntax	Example
Less than	`<`	`x < y`
Greater than	`>`	`z > 1`
Less than or equal to	`<=`	`b <= 1`
Greater than or equal to	`>=`	`c >= 2`

Equality operators	Syntax	Example
Equal to	`==`	`a==b`
Not equal to	`!=`	`b!=3`


IDLE Shel...

```
>>> 13<30
    True
>>> 13>30
    False
>>> 13<=30
    True
>>> 13>=30
    False
>>> 13==30
    False
>>> 13!=30
    True
```

Ln: 19 Col: 0

NOTE: A<B<C

- In python, you may test a variable in certain range like this
  - 1 > a > 4
  - 4 < b <12
- Not all programming languages support this expression, e.g., C++ and Java do not support this syntax and should use `a < b && b < c` instead of `a < b < c`

```
min=10
max=15
a=int(input("Please enter an integer: "))
if min < a < max:
    print("You hit the Jackpot!")
else:
    print("Sorry, please try again")
```

PRECEDENCE AND ASSOCIATIVITY

- Precedence: The order of evaluation when an expression consists of multiple operators
- Associativity: The order of evaluation on operators with same precedence

PRECEDENCE & ASSOCIATIVITY OF OPERATORS, AGAIN

Operator precedence (high to low)	Description	Associativity
()	Parentheses	Left to right
**	Exponent	Right to left
+, -	Unary plus, Unary minus	Left to right
*, /, //, %	Multiplication, Division, Floor division, Modulus	Left to right
+, -	Addition, Subtraction	Left to right
==, !=, >, >=, <=	Comparisons, Identity	
not	Logical NOT	
and	Logical AND	
or	Logical OR	

SUMMARY

- A logical expression is either `True` or `False`
- Conditional statements are statements that will only execute under certain condition.
- Keyword: `if`, `elif`, `else`

```
if logical_expression_1:

→statement
→statement

elif logical_expression_2:

→statement
→statement


else:

→statement
→statement
```

SUMMARY

- Writing conditional statement is not difficult, difficult is make it right
  - Right condition test
  - Right statement for true case and false case