

# Conditional Execution

Decision making with `if-else` statements

Alwin Tareen

# Boolean Expressions

## The Boolean Data Type

- ▶ A **boolean expression** is an expression that is evaluated to either True or False.
- ▶ In Python, the boolean data type can only take the values True or False.
- ▶ Note that True or False must begin with a capital letter. These are special values which are not strings.
- ▶ Python specifies this data type as: `bool`

# Comparison Operators

- ▶ The following comparison operators compare two numbers, and give back a boolean value.

Comparison Operator	Description
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

- ▶ Note that == denotes logical equality, while = is the assignment operator.
- ▶ Confusing these two items is a common source of errors.

# Logical Operators

- ▶ There are three logical operators in Python.

Rank	Operator	Example	Result
1	not	not a	True if a is False, and False if a is True.
2	and	a and b	True if a and b are both True, and False otherwise.
3	or	a or b	True if either a or b are True, and False otherwise.

- ▶ Logical operators must be evaluated in the following order of precedence: not, and, or.

# Truth Tables

- ▶ A logical operation can be described by a **truth table** that lists all of the possible combinations of values for the input variables involved in an expression.
- ▶ The following is a two-valued truth table. It shows the outputs for the and, or operators.

		logical and	logical or
a	b	a and b	a or b
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

# The not Operator

- ▶ The not operator gives the logical complement of a boolean value.
- ▶ It does not alter the variable upon which it acts.
- ▶ The following is the truth table for the not operator:

a	logical not
	not a
False	True
True	False

```
if (not lights):  
    print("The room is dark.")
```

# The and Operator

- ▶ The result of a logical and operation is True if both operands are True, but False otherwise.

```
if (chips > 0 and soda > 0):  
    print("You have snacks.")
```

# The or Operator

- ▶ The result of a logical or operation is True if one or the other or both of the operands are True, but False otherwise.

```
if (money > 1000 or creditcard == True):  
    print("You can buy an iPhone.")
```



# The if Statement

## Conditional Execution

Conditional statements give us the ability to check a condition, and change the behaviour of the program accordingly.

## The if Statement

- ▶ This allows us to perform actions only when certain conditions are met, and to skip the block of code otherwise.
- ▶ The structure of the if statement consists of the following:

```
if condition:  
    code block of statements
```

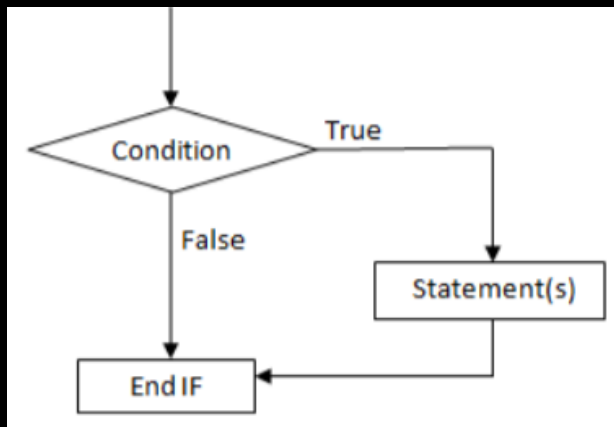
# The if Statement

- ▶ The first line has the keyword `if`, then a **condition** which must be True or False expression, then a colon.
- ▶ Then, there is the body of the `if` statement, which consists of one or more indented lines.
- ▶ According to good programming practice, there should be 4 spaces of indent.

```
if age >= 18:  
    print("You can drive.")
```

- ▶ If the logical condition is True, then the indented block of code gets executed. If the logical condition is False, then the indented block of code is skipped.

# Flowchart of the if Statement



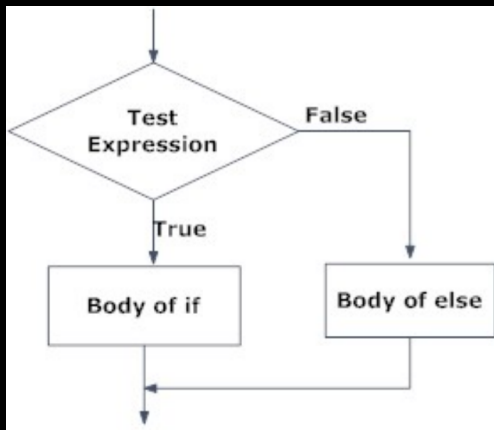
# The if-else Statement

- ▶ Often, you want to test some condition, and take either one action or another action, depending upon whether the condition is True or False.
- ▶ With an if-else statement, there are two possibilities, and the condition determines which one is executed.
- ▶ The structure of the if-else statement consists of the following:

```
if condition:  
    code block if the condition is True  
else:  
    code block if the condition is False
```

# Flowchart of the if-else Statement

- ▶ Since the condition must be either True or False, exactly one of the possible branches must be executed.



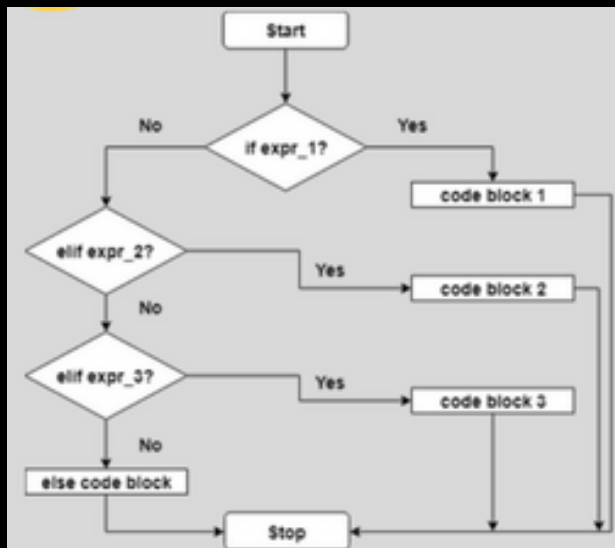
# The `if-elif-else` Statement

- ▶ Sometimes, there are more than two possibilities which can be selected, so we need more than two logical branches.
- ▶ We can use an `if-elif-else` statement, which allows us to check several conditions in a row.
- ▶ The keyword `elif` is an abbreviation for `else if`, since it is the same as putting an `if` statement inside an `else` block.
- ▶ You can combine any number of `elif` statements in your structure.
- ▶ The `else` statement at the end is optional. If you include an `else` statement, then it must come at the very end of the structure.

# The if-elif-else Statement

```
if first_condition:  
    code block if first_condition is True  
elif second_condition:  
    code block if second_condition is True  
elif third_condition:  
    code block if third_condition is True  
else:  
    code block if all other conditions are False
```

# Flowchart of the if-elif-else Statement





# Short-Circuit Evaluation

- ▶ The `and` and `or` operators are short-circuited.
- ▶ If the left operand is enough to decide the boolean result of the operation, then the right operand is not evaluated.

## `and`

If the left operand is `False`, then the result of the entire expression will be `False`, no matter what the right operand is.

```
sh = False and (False and (True or False) or True)
```

## `or`

If the left operand is `True`, then the result of the entire expression will be `True`, no matter what the right operand is.

```
cr = True or (False and (True or False) or True)
```

# Conditionals: End of Notes