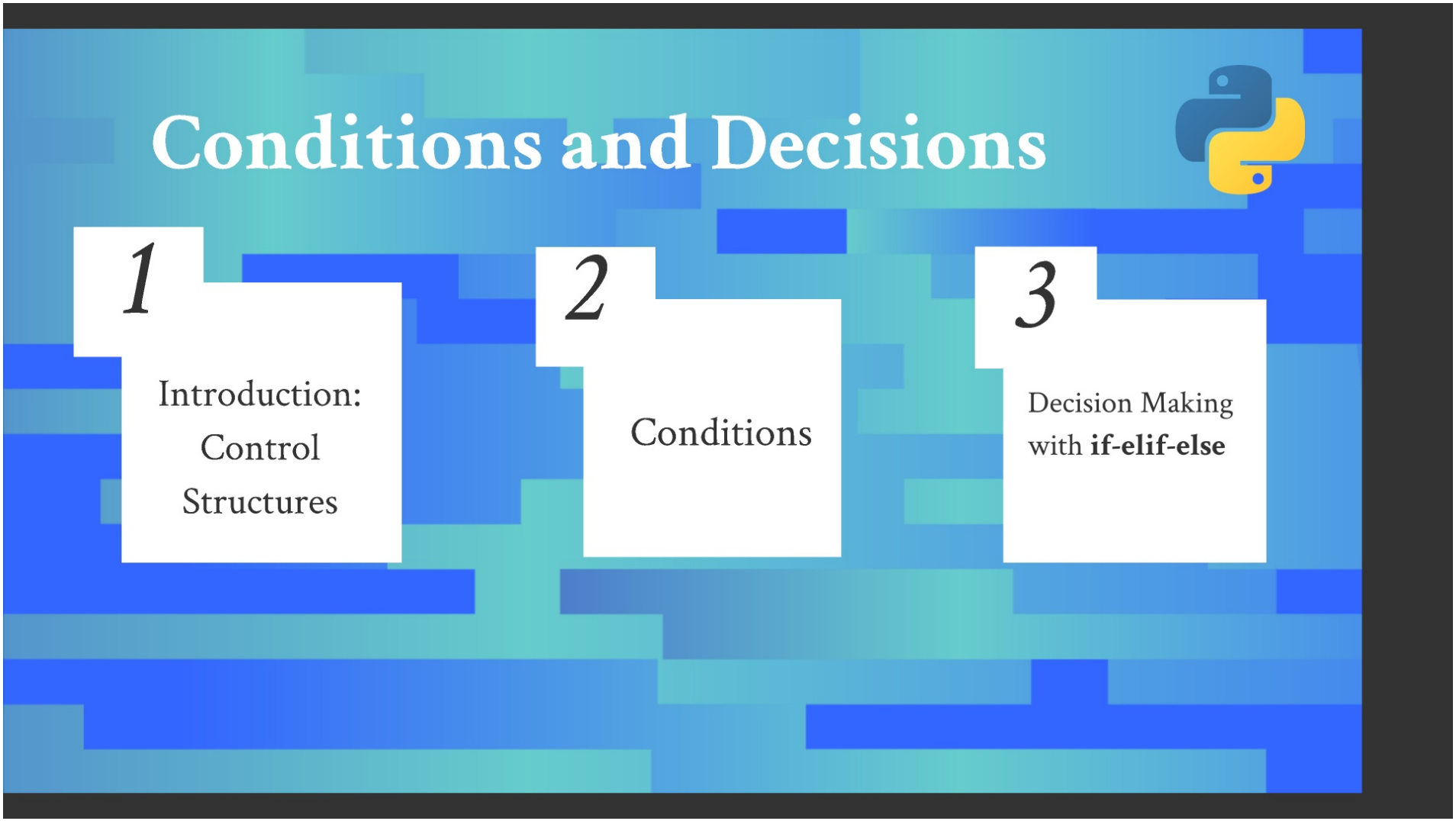
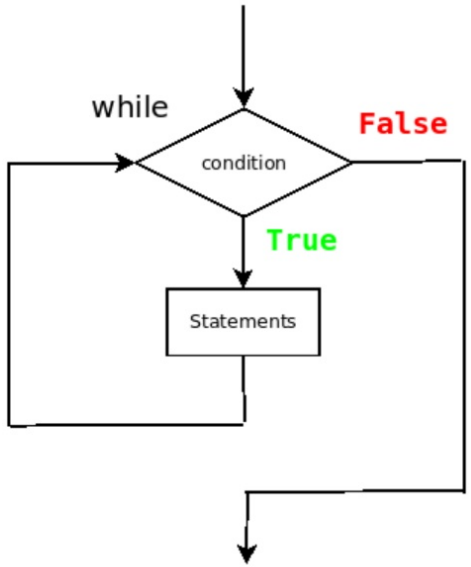

Conditions and Decisions
1
Introduction: Control Structures
2
Conditions
3
Decision Making with **if-elif-else**

Python's Control Structures

Decision Making: if-elif-else

Repetition: while

Iteration: for



Introduction

Decision Making with **if-elif-else**

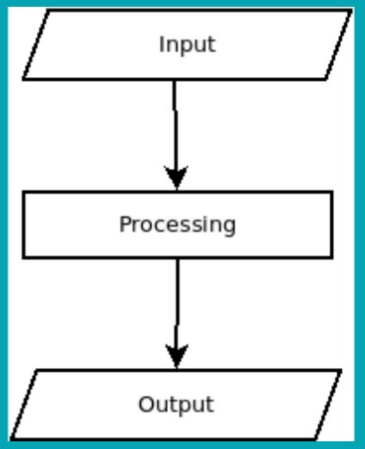
Repetition with **while**

Iteration with **for**

Introduction to Control Structures

The previous programs all followed the same format, involving a sequence of statements, executed one after another, from top to bottom.

```
# Input metres
metres = float(input("Enter the distance in metres: "))

# Calculate feet as metres x 3.28084
feet = metres * 3.28084

# Calculate whole_feet as integer part of feet
whole_feet = int(feet)

# Calculate inches as (feet - whole_feet) * 12
inches = (feet - whole_feet) * 12

# Print whole_feet, inches
print(f"Equivalent distance is {whole_feet} feet, {inches:.1f} inches")
```



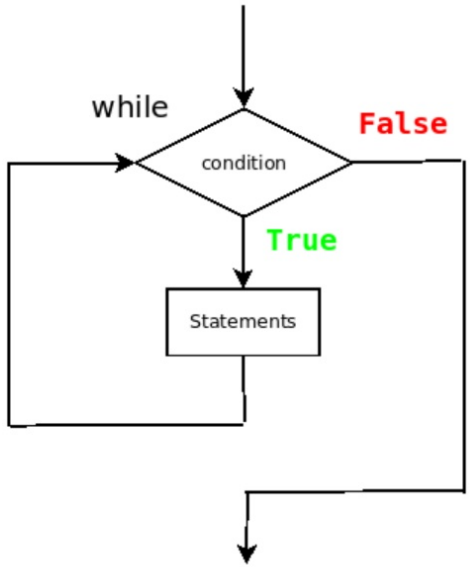
Control structures provide a mechanism to control the flow of statement execution.

Python's Control Structures

Decision Making: if-elif-else

Repetition: while

Iteration: for



Introduction

Decision Making with **if-elif-else**

Repetition with **while**

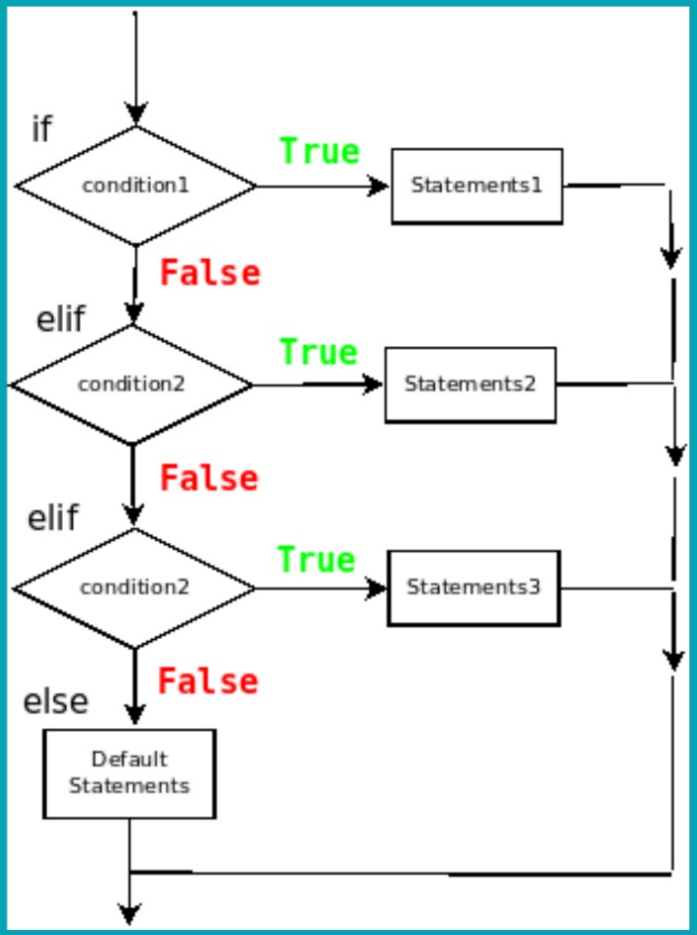
Iteration with **for**

Decision Making with **if-elif-else**

This is a **decision making** structure, also called a *choice*, *selection* or *branching* structure.

```
if condition1:
   statements1
elif condition2:
   statements2
elif condition3:
   statements3
else:
   default statements
```

The program chooses between two or more alternatives (statement blocks), depending on the evaluation of one or more **boolean** conditions.

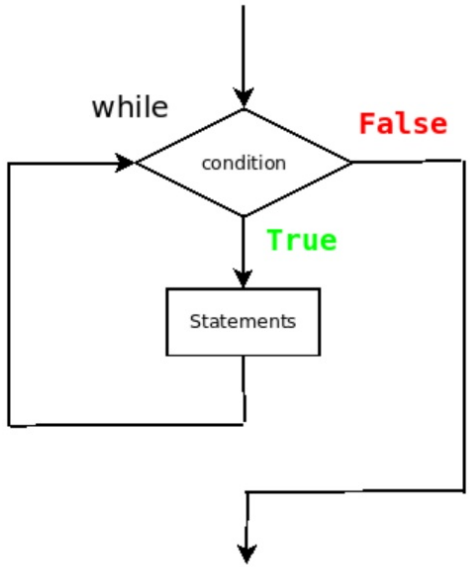


Python's Control Structures

Decision Making: if-elif-else

Repetition: while

Iteration: for



Introduction

Decision Making with **if-elif-else**

Repetition with **while**

Iteration with **for**

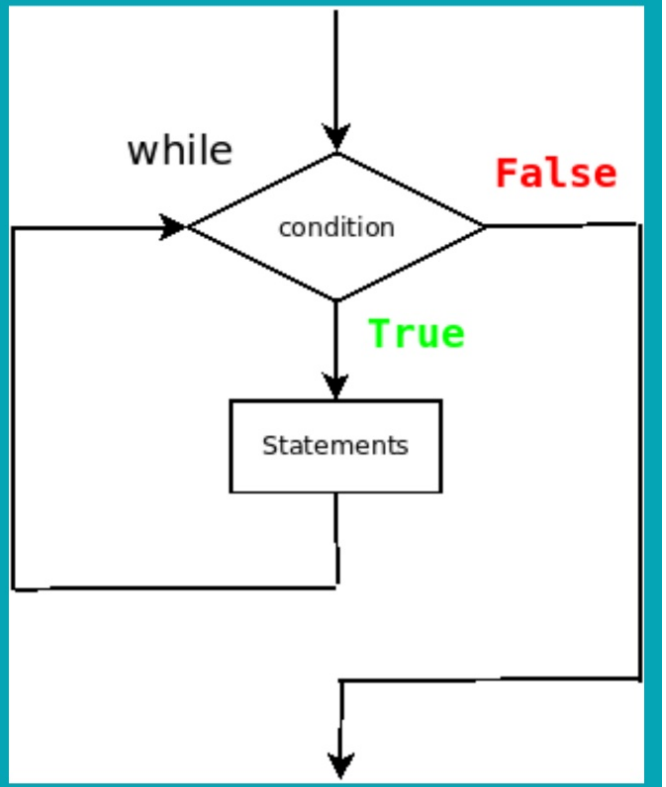
Repetition with **while**

This is a *repetition* structure

```
while condition:
    statement(s)
```

The program *repeats* a block of statements **while** a **boolean** condition evaluates as **True**.

That is, the loops *repeats* as long as the **boolean** condition is **True**.

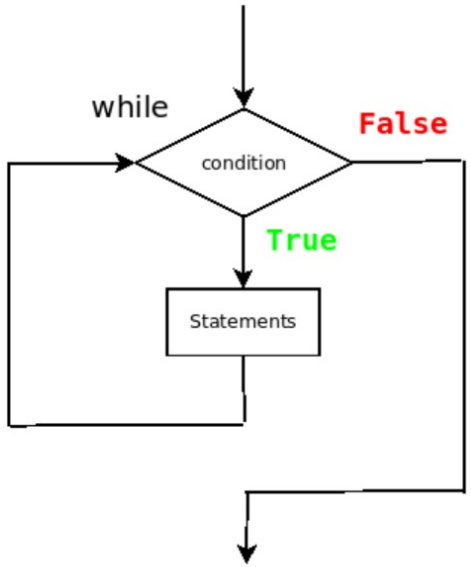


Python's Control Structures

Decision Making: if-elif-else

Repetition: while

Iteration: for



Introduction

Decision Making with **if-elif-else**

Repetition with **while**

Iteration with **for**

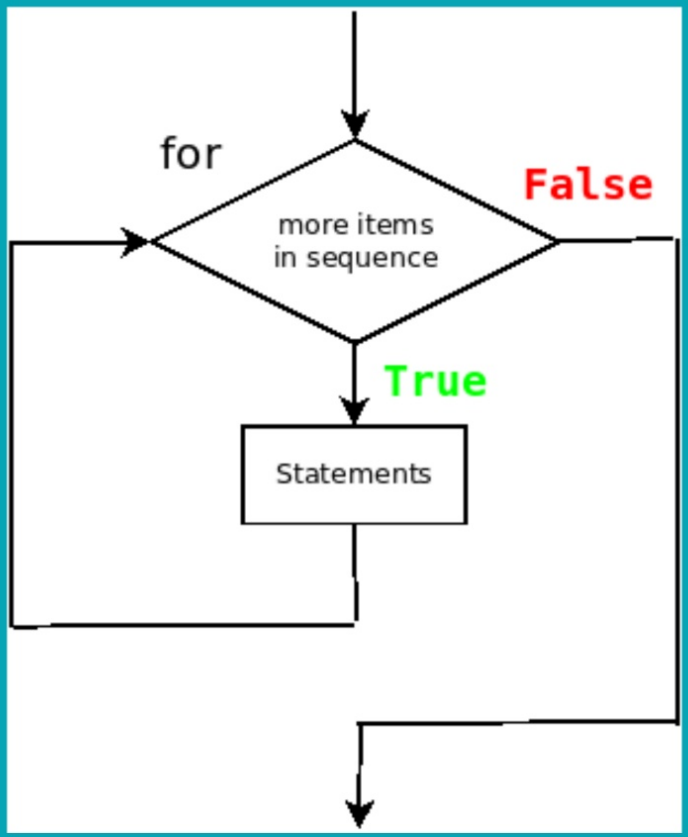
Iteration with **for**

This is another repetition structure.

```
for variable in sequence:
    statement(s)
```

The program repeats a block of statements for each value in the sequence provided.

It does this by *iterating* through the values in the sequence, on at a time.

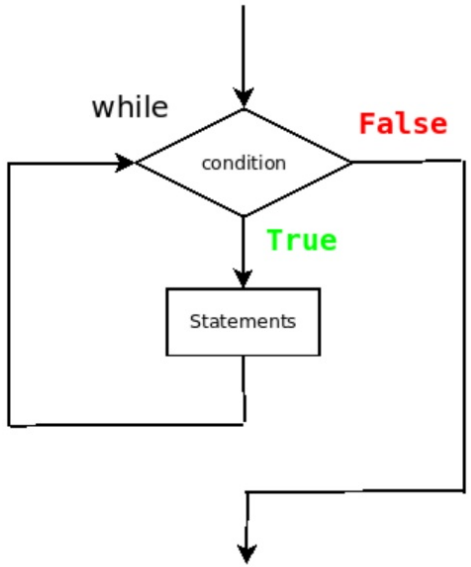


Python's Control Structures

Decision Making: if-elif-else

Repetition: while

Iteration: for

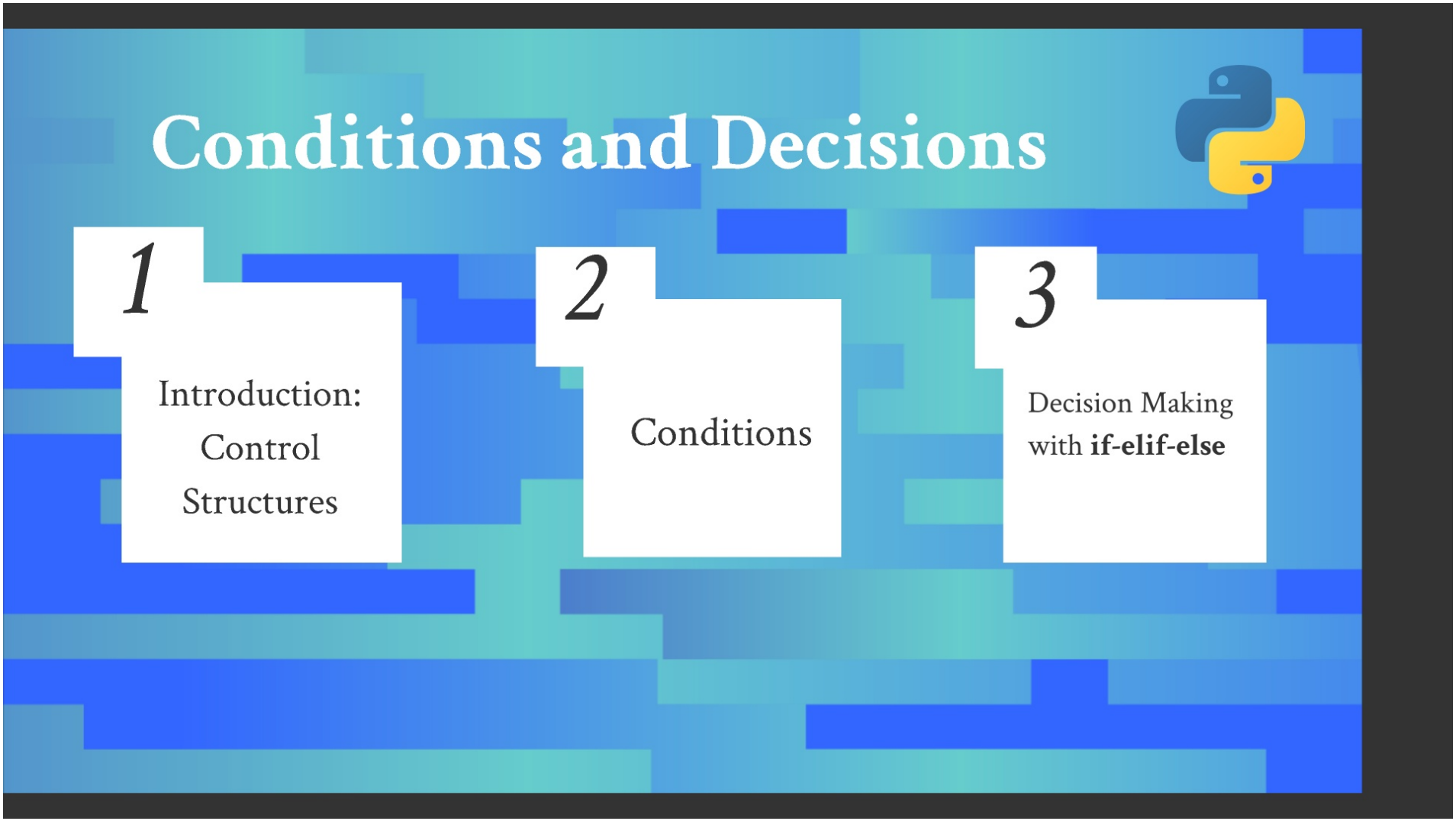


Introduction

Decision Making with **if-elif-else**

Repetition with **while**

Iteration with **for**

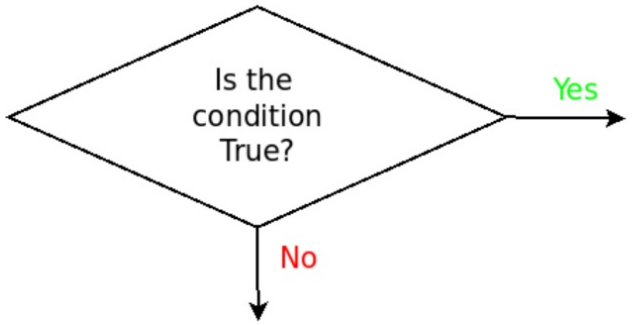

Conditions and Decisions
1
Introduction: Control Structures
2
Conditions
3
Decision Making with **if-elif-else**

Conditions

Also known as Conditional Expressions / Boolean Expressions

Used in decision making with **if-elif-else** structures and **while** repetition loops.

When evaluated, the result of a condition is either:
True meaning the condition holds
or **False** meaning the condition does not hold.



Conditions are questions that have a yes/no answer:
"Yes" is represented in Python as **True**
"No" is represented in Python as **False**

Conditions using Relational Operators

Conditions involving simple comparisons have the following form:

	value 1	***relational* operator**	**value 2**
e.g.	temperature	>	27

A value can be:

- a simple value, such as 0, "a", 38.6, "Hello World"
- the value of a variable, such as **temperature**
- an *expression* (combination of values, operators and variables), which must be evaluated, e.g.

9.0/5 * celsius + 32

Relational Operators

Used in conditions to evaluate the relationship between two values:

Operator	Meaning	Example
<	Less Than	3 < 5 returns True and 5 < 3 gives False
>	Greater Than	5 > 3 returns True.
<=	Less Than or Equal To	x = 3; y = 6; x <= y returns True.
>=	Greater Than or Equal To	x = 4; y = 3; x >= 3 returns True.
==	Equal To	x = 2; y = 2; x == y returns True.
!=	Not Equal To	x = 2; y = 3; x != y returns True.

Examples

Temperature above 27 degrees

```
In [3]: temperature = 28

In [4]: temperature > 27
Out[4]: True
```

```
In [1]: temperature = 18

In [2]: temperature > 27
Out[2]: False
```

Bank Account in Credit

```
In [30]: bank_balance = 135

In [31]: bank_balance > 0
Out[31]: True
```

```
In [32]: bank_balance = -742

In [33]: bank_balance > 0
Out[33]: False
```

Money to spend on Credit Card

```
In [27]: credit_balance = 500

In [28]: credit_limit = 1000

In [29]: credit_balance < credit_limit
Out[29]: True
```

```
In [34]: credit_balance = 1500

In [35]: credit_limit = 1000

In [36]: credit_balance < credit_limit
Out[36]: False
```

Another example

Free space less than 10% of total space

```
In [24]: total = 500

In [25]: used = 473

In [26]: (total - used) / total < 0.1
Out[26]: True
```

```
In [9]: total = 500

In [10]: used = 350

In [11]: (total - used) / total < 0.1
Out[11]: False
```

Boolean Functions/Methods as Conditions

Boolean functions/methods return **True** or **False**

They can be used as conditional expressions.

Examples: String methods

```
student_id.startswith("A00")

email_address.endswith("@ait.ie")

username.islower()
```

```
In [1]: student_id = "A00123456"

In [2]: student_id.startswith("A00")
Out[2]: True

In [3]: student_id = "87014220"

In [4]: student_id.startswith("A00")
Out[4]: False
```

```
In [10]: username = "jbloggs"

In [11]: username.islower()
Out[11]: True

In [12]: username = "JBloggs"

In [13]: username.islower()
Out[13]: False
```

Conditional Operators **and** and **or**

Used to combine two conditions and produce a Boolean (True/False) result.

condition 1 **and** condition 2
The **and** of two conditions is true exactly when *both* of the conditions are **True**.

condition 1 **or** condition 2
The **or** of two conditions is true when *either* condition is **True** (or both conditions).

P	*Q*	*P and Q*
T	T	T
T	F	F
F	T	F
F	F	F

P	*Q*	*P or Q*
T	T	T
T	F	T
F	T	T
F	F	F

and Example

A program is to process a student's mark out of a hundred; the logic to check for a valid mark is:

greater than or equal to 0 AND less than or equal to 100

mark	mark >= 0	mark <= 100	valid mark
75	True	True	True
110	True	False	False
-25	False	True	False

The condition is:

mark >= 0 **and** mark <= 100

Python allows the following syntax:

0 <= mark <= 100

Other Examples

No money in the bank and credit card maxed out!

```
bank_balance <= 0 and credit >= limit
```

You can combine any number of **and**s:

Healthy cholesterol levels

```
total <= 5 and ldl <= 3 and hdl > 1
```

or Examples

Amusement parks have age and height restrictions for their rides:
age < 3 **or** height < 1.02 not permitted on ride

If you're under 14 or weigh more than 235kg, you can't bungee jump:
age < 14 **or** weight > 235 not permitted to jump

or Examples expressed using and

Some amusement parks have age and height restrictions for their rides,
age >= 3 **and** height >= 1.02 permitted on ride

If you're 14 or over and weigh no more than 235kg, you *can* bungee jump:
age >= 14 **and** weight <= 235 permitted to jump

Combining Conditions with **and** and **or**

You can use these operators together to make complex conditions.

The interpretation of the conditional expressions relies on the *precedence* rules for the operators.

The rule is:

and is evaluated before **or**

unless brackets are used to override this

Example: mature or part-time engineering students

There's a difference between:

age > 23 or status == 'part-time' and school = 'engineering' 1

and

school = 'engineering' and age > 23 or status == 'part-time' 2

age	status	school	1	2
25	part-time	engineering	True	True
25	full-time	engineering	True	True
18	part-time	engineering	True	True
18	full-time	engineering	False	False
18	part-time	science	False	True
25	full-time	science	True	False

If unsure, use brackets ( ) in a combined condition involving **and** and **or**.

The not operator

The **not** operator yields the opposite of a Boolean expression: It reverses the "truth" of a condition.

P	*not P*
T	F
F	T

If a Boolean Expression P is **True** then **not** P is **False**.

If a Boolean Expression P is **False** then **not** P is **True**.

Examples with String Methods:

This is useful for input validation, e.g. ensuring a valid AIT student ID has been input:

not student_id.startswith("A00")

```
In [1]: student_id = "A00123456"

In [2]: student_id.startswith("A00")
Out[2]: True

In [3]: student_id = "87014220"

In [4]: student_id.startswith("A00")
Out[4]: False
```

```
In [6]: student_id = "A00123456"

In [7]: not student_id.startswith("A00")
Out[7]: False

In [8]: student_id = "87014220"

In [9]: not student_id.startswith("A00")
Out[9]: True
```

Similarly, you can check if any of the letters in a username are not lowercase (in which case the username would be invalid) using:

not username.islower()

```
In [10]: username = "jbloggs"

In [11]: username.islower()
Out[11]: True

In [12]: username = "JBloggs"

In [13]: username.islower()
Out[13]: False
```

```
In [14]: username = "jbloggs"

In [15]: not username.islower()
Out[15]: False

In [16]: username = "JBloggs"

In [17]: not username.islower()
Out[17]: True
```

More Falseness

Python lets you use any value where it expects a Boolean value, for example with **if-elif** and **while**.

The following all evaluate as **False**:

- **None** (represented in C/C++/Java as null)
- An empty string ""
- The number zero: 0 or 0.0
- An empty data structure: list [] tuple () or dictionary {}

Everything else evaluates as **True**.

Using None or empty values as False

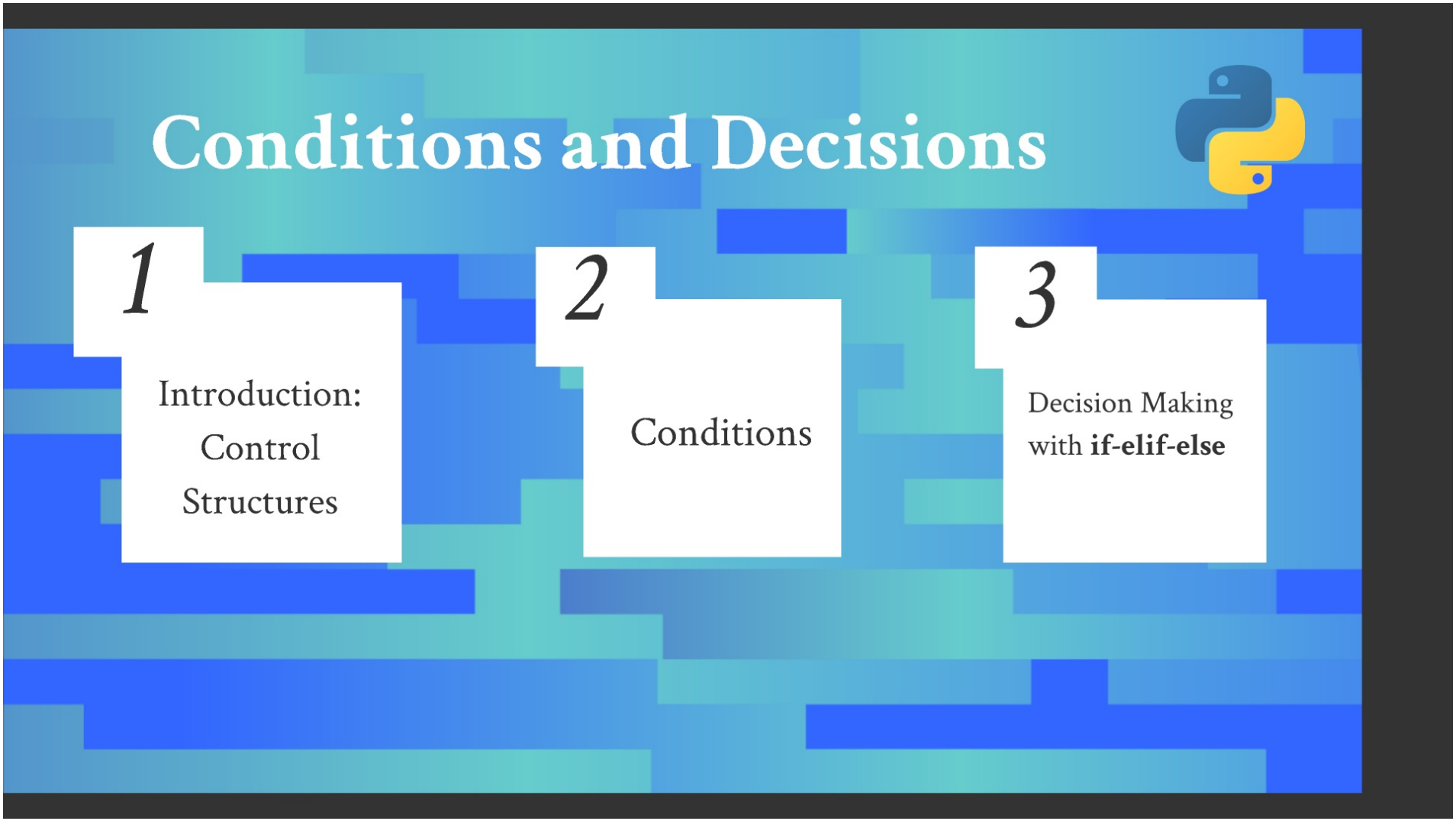
This makes it easy to check for null values (**None**), empty strings, or empty data structures.

For example, you could check if a user has not provided any input in response to a prompt:

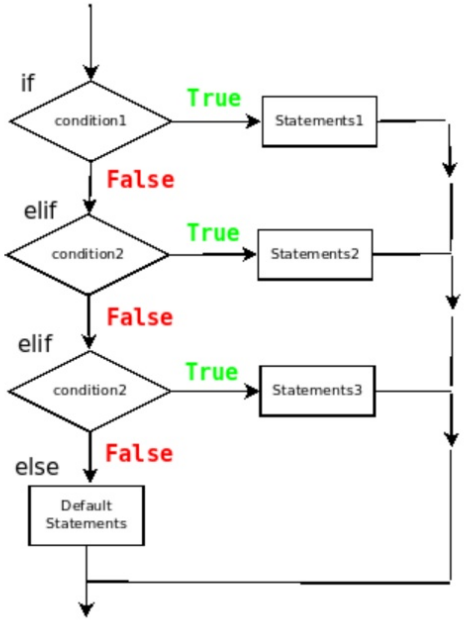
```
name = input("Enter your name: ")

if name: # name is not an empty string
    print("Welcome", name)
else:
    print("You didn't enter anything!")
```

Output

```
Enter your name:
You didn't enter anything!
```


Conditions and Decisions
1
Introduction: Control Structures
2
Conditions
3
Decision Making with **if-elif-else**

Decision Making with **if-elif-else**



1

```
if condition:
    statements
```

2

```
if condition:
    statements1
else:
    statements2
```

3

```
if condition1:
    statements1
elif condition2:
    statements2
else:
    statements3
```

4

Additional Topics

Decision Making with if-elif-else

Examples of Decisions

Program	*Decision*
ATM	Has a card been entered?
	Does the PIN match the card?
Windows	Has the user logged in correctly?
Spyder	Has the user clicked on Save?
Fridge	Is the door open more than 3 seconds?

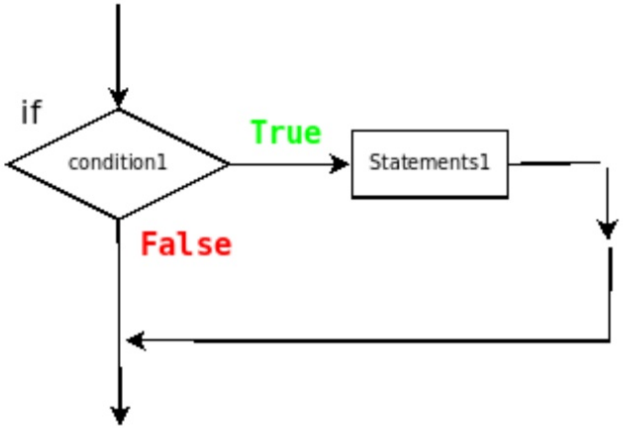
Decision making structures allow a program to execute different instructions for different cases. This allows the program to "choose" an appropriate course of action, depending on the situation.

Python's *only* decision making structure is **if-elif-else** which chooses between two or more alternatives (statement blocks), depending on the evaluation of one or more boolean conditions.

One-Way Decisions with if

Python uses **if** statements to implement decisions.

A "One-Way" Decision is required when the program needs to take an action, (i.e. execute one or more statements,) when a condition evaluates as **True**, and otherwise the program takes no action.



The syntax is

if ***condition*****:**
    statement(s)

statement(s) are a sequence of one or more statements indented under, and associated with, the **if** statement.

The ***condition*** is a check to see if these statements should be executed.

The Importance of indentation in Python

Python uses *indentation* (spaces or tabs) to signify blocks of code, rather than braces {}.

The line containing the **if** must end with a colon **:** indicating that an indented block is to follow.

if *condition*:
⟶ *statement(s)*

De-denting (unindenting) signifies the end of the block.
https://unspecified.wordpress.com/2011/10/18/why-pythons-whitespace-rule-is-right/

Example: Checking for a high temperature

A program is required which

- inputs a temperature value (in degrees Celsius)
- displays a "Status Yellow Warning" message if the temperature is above 27

Sample Values

Input: temperature	Output
18	
27	
28	"Status Yellow Warning"

Specification Table

Input	Processing	Output
temperature	Input temperature If temperature > 27 Print message	message

Program

```
# Program to display a high temperature warning, if appropriate

# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# If temperature > 27
if temperature > 27:
    # Print message
    print("Status Yellow Warning")
```

Sample Output from executing the program 3 separate times.

```
Enter Celsius Temperature: 18

In [2]:
```

No output

```
Enter Celsius Temperature: 27

In [3]:
```

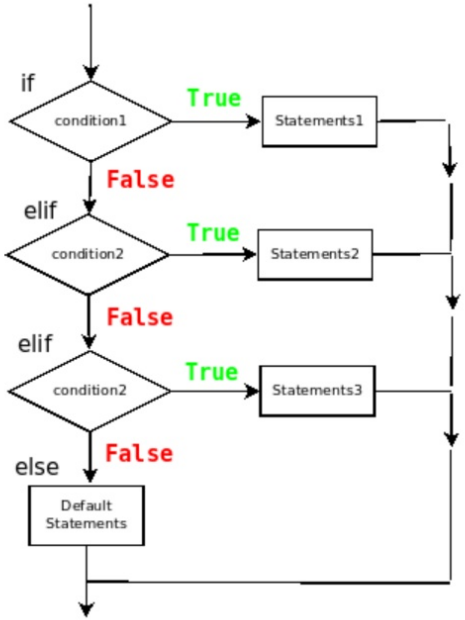
No output

```
Enter Celsius Temperature: 28
Status Yellow Warning
```

Output

Input	Output		Pass Y/N?
	Expected	Actual	
18	(no output)	(no output)	Y
27	(no output)	(no output)	Y
28	Status Yellow Warning	Status Yellow Warning	Y

Decision Making with **if-elif-else**



1

```
if condition:
    statements
```

2

```
if condition:
    statements1
else:
    statements2
```

3

```
if condition1:
    statements1
elif condition2:
    statements2
else:
    statements3
```

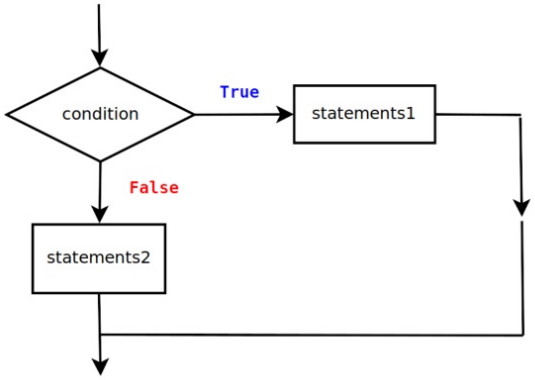
4

Additional Topics

Two-Way Decisions with if-else

A "Two-Way" Decision involves a situation where the program needs to perform one action (i.e. execute one or more statements) if a condition evaluates as **True**, otherwise the program will perform a different action (a separate block of statements).

It is represented using **if-else**, and is implemented by attaching an **else** clause onto an **if** clause.




```
if condition:
    statements1
else:
    statements1
```

Notice the colon **:** at the end of the **else** line.

As with the colon at the end of the **if** line, this must be included as it indicates an associated block of code, which must also be *indented*.

Example: Temperature Warnings continued

The previous program can be extended to display a "Status Green" message if the temperature is not over 27 degrees Celsius.

Sample Values

Input: temperature	Output
18	"Status Green"
27	"Status Green"
28	"Status Yellow Warning"

Specification Table

Input	Processing	Output

temperature	Input temperature
	If temperature > 27
	  Print Status Yellow message
	Else
	  Print Status Green message
		message

Program

```
# Program to display the high temperature status

# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# If temperature > 27
if temperature > 27:
    # Print message
    print("Status Yellow Warning")
# Otherwise
else:
    print("Status Green")
```

Sample Outputs

```
Enter Celsius Temperature: 18
Status Green
```

```
Enter Celsius Temperature: 27
Status Green
```

```
Enter Celsius Temperature: 28
Status Yellow Warning
```

Testing

Input	Output		Pass Y/N?
	Expected	Actual	
18	Status Green	Status Green	Y
27	Status Green	Status Green	Y
28	Status Yellow Warning	Status Yellow Warning	Y

Alternative Version: status variable

Uses an **if-else** structure to determine the message to store in a variable, **status**.

After the **if-else**, the message is displayed, by displaying the contents of the variable.

Input	Processing	Output

temperature	Input temperature
	If temperature > 27
	  Set status to Yellow
	Else
	  Set status to Green
	Print status
		status

```
# Program to display high temperature status

# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# If temperature > 27
if temperature > 27:
    status = "Yellow"
else:
    status = "Green"

# Print status
print("Temperature Status:", status)
```

Python's Ternary Conditional Operator

A simple **if-else** can be expressed in one line using Python's Ternary Conditional Operator.

The syntax is:

value1 **if** *condition* **else** *value2*

Explanation:

If the *condition* is **True**
value1 is used
otherwise
value2 is used

For Example:

status = "Yellow" **if** temperature > 27 **else** "Green"

status = “Yellow” **if** temperature > 27 **else** “Green”

If

```
In [18]: temperature = 28

In [19]: status = "Yellow" if temperature > 27 else "Green"

In [20]: status
Out[20]: 'Yellow'
```

Else

```
In [21]: temperature = 18

In [22]: status = "Yellow" if temperature > 27 else "Green"

In [23]: status
Out[23]: 'Green'
```

Program is slightly shorter

```
# Program to display high temperature status

# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# Set the status
status = "Yellow" if temperature > 27 else "Green"

# Print status
print("Temperature Status:", status)
```

You can use this in a **print** function, which makes the program even shorter:

```
# Program to display high temperature status

# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# Print the status
print("Status: Yellow" if temperature > 27 else "Status: Green")
```

Points to consider with the ternary operator:

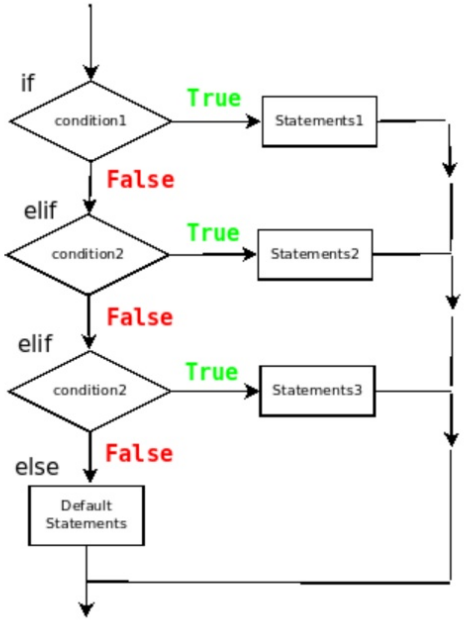
- There are fewer lines of code
- There is no difference in performance compared with a classic **if-else**
- The order is different from the corresponding structure in traditional programming languages

<condition> ? <expression1> : <expression2>

- Debugging the classic **if-else** is easier
- The syntax could be misleading (precedence rules)

https://blog.softhints.com/python-3-if-else-one-line-or-ternary-operator/

Decision Making with **if-elif-else**



1

```
if condition:
    statements
```

2

```
if condition:
    statements1
else:
    statements2
```

3

```
if condition1:
    statements1
elif condition2:
    statements2
else:
    statements3
```

4

Additional Topics

Multi-way Decisions with if-elif-else

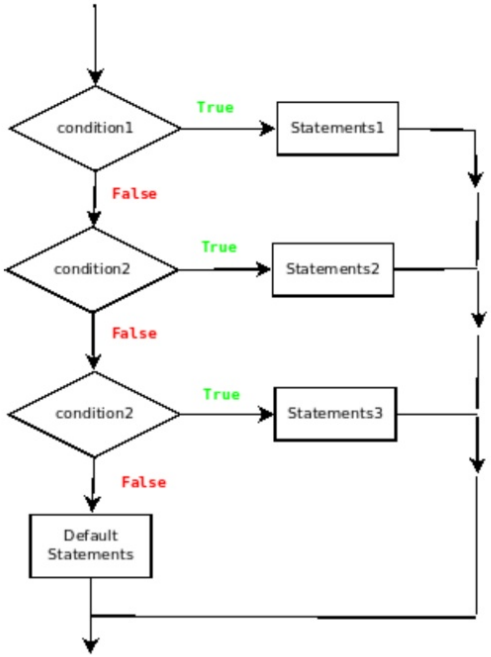
A Multi-Way Decision involves more than two possible alternatives.

The program needs to execute one or more statements if the first condition is **True**, otherwise it will perform a different block of statements, if the second condition (or some further condition) evaluates as **True**,
otherwise the program will take some specified default action.

It is represented using **if-elif-else**

```
if condition1:
    statements1
elif condition2:
    statements2
elif condition3:
    statements3
else:
    default statements
```

elif represents "Else If".



Example: Cholesterol Levels

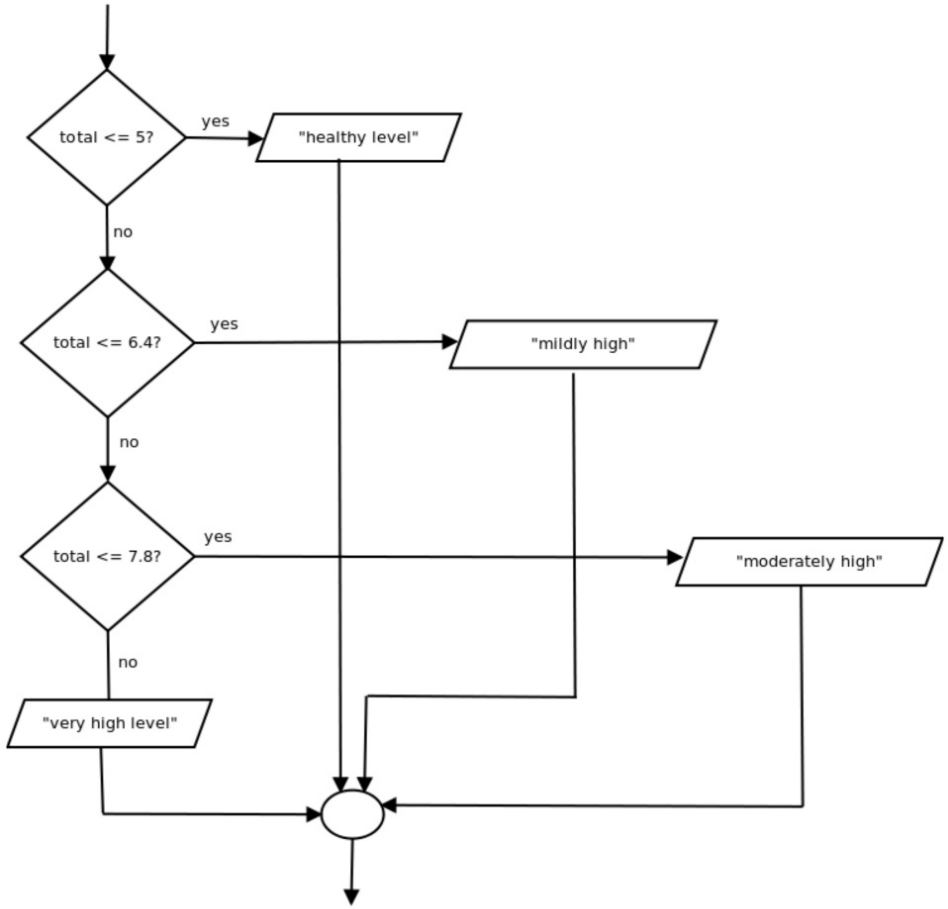
High Cholesterol levels may lead to heart disease. There are two types of Cholesterol:

- HDL (high density lipoprotein) "good" cholesterol
- LDL (low density lipoprotein) "bad" cholesterol

A diagnosis of high cholesterol is based on calculating the total (HDL+LDL) cholesterol:

Total Cholesterol	Description
0 up to 5	Healthy cholesterol level
Above 5 and up to 6.4	Mildly high cholesterol level
Above 6.4 and up to 7.8	Moderately high level
Above 7.8	Very high cholesterol

This requires a *multi-way* decision structure.



The logic is as follows:

- If total is 5 or less, then display "healthy cholesterol", and it's finished.
- Otherwise (so the total is greater than 5), if the total is 6.4 or less (and greater than 5), then display "mildly high level"
- Otherwise (so the total is more than 6.4), if the total is 7.8 or less (and greater than 6.4), then display "moderately high level".
- Otherwise (so the total is more than 7.8), display "very high level".

Python Implementation

```
if total <= 5:
    print("Healthy cholesterol level")
elif total <= 6.4:
    print("Mildly high cholesterol level")
elif total <= 7.8:
    print("Moderately high cholesterol level")
else:
    print("Very high cholesterol level")
```

Common Mistakes

The **elif** *must* have a condition associated with it.

```
23 # determine and display message
24 if total <= 5:
25     print("Healthy cholesterol level")
26 elif total <= 6.4:
27     print("Mildly high cholesterol level")
28 elif total <= 7.8:
29     print("Moderately high cholesterol level")
30 elif:
31     print("Very high cholesterol level")
```

```
    elif:
       ^
SyntaxError: invalid syntax
```

The **else** must *not* have a condition associated with it.

```
23 # determine and display message
24 if total <= 5:
25     print("Healthy cholesterol level")
26 elif total <= 6.4:
27     print("Mildly high cholesterol level")
28 elif total <= 7.8:
29     print("Moderately high cholesterol level")
30 else total > 7.8:
31     print("Very high cholesterol level")
```

```
    else total > 7.8:
              ^
SyntaxError: invalid syntax
```

Processing Values in Order

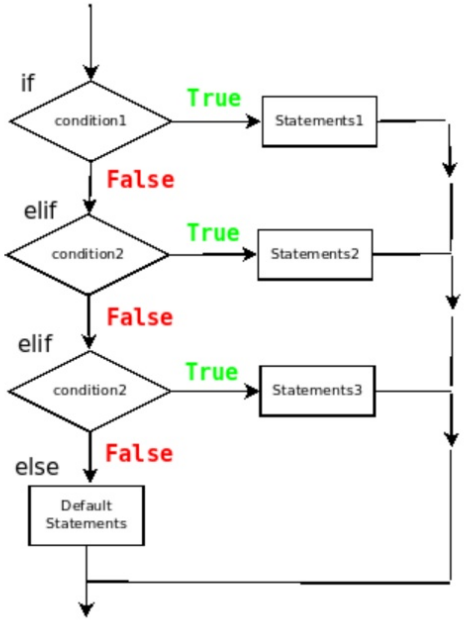
The **if-elif-else** processed the cholesterol values in order, lowest to highest:

```
if total <= 5:
    print("Healthy cholesterol level")
elif total <= 6.4:
    print("Mildly high cholesterol level")
elif total <= 7.8:
    print("Moderately high cholesterol level")
else:
    print("Very high cholesterol level")
```

Alternatively, you could process the values in order, highest to lowest:

```
if total > 7.8:
    print("Very high cholesterol level")
elif total > 6.4:
    print("Mildly high cholesterol level")
elif total > 5:
    print("Moderately high cholesterol level")
else:
    print("Healthy cholesterol level")
```

Decision Making with **if-elif-else**



1

```
if condition:
    statements
```

2

```
if condition:
    statements1
else:
    statements2
```

3

```
if condition1:
    statements1
elif condition2:
    statements2
else:
    statements3
```

4

Additional Topics

Decision Making with Strings

For example, the username for a specific Social Media account is required to have no more than 15 characters.

The following program inputs a username and then displays a message indicating whether or not the length of the username is suitable.

```
# Input username
username = input("Enter the username: ")

# Check if the length is valid
if len(username) <= 15:
    print("Username length is acceptable")
else:
    print("Username is too long")
```

Sample Outputs

```
Enter the username: joebloggs
Username length is acceptable
```

```
Enter the username: joebloggsisthegreatest
Username is too long
```

The **len**() function (**len**gth) checks the number of characters in the string.

Another possible restriction on a username is that the characters must not contain any uppercase letters.

This can be checked for using the string method **islower**(), which returns **True** if all letters in the string are lowercase, **False** otherwise (i.e. if there are any uppercase characters - or if there are no lowercase letters at all).

```
# Input username
username = input("Enter username: ")

# Check if the username is lowercase
if username.islower():
    print("Username is acceptable")
else:
    print("Username contains uppercase letter(s)")
```

Sample Outputs

```
Enter the username: joebloggs
Username is acceptable
```

```
Enter the username: Joebloggs
Username contains uppercase character(s)
```

```
Enter the username: joebloggs123
Username is acceptable
```

Note that an incorrect message is displayed if there are no lowercase letters in the username string:

```
Enter the username: 123456
Username contains uppercase character(s)
```

This is because the **islower**() method returns **False** if there are no lowercase letters in the string.

A correct implementation of username validation will be presented in Section 2(b) "Python's **for** loop".

Nested Ifs

A block of code corresponding to an **if**, **elif** or **else** can contain any valid statements.

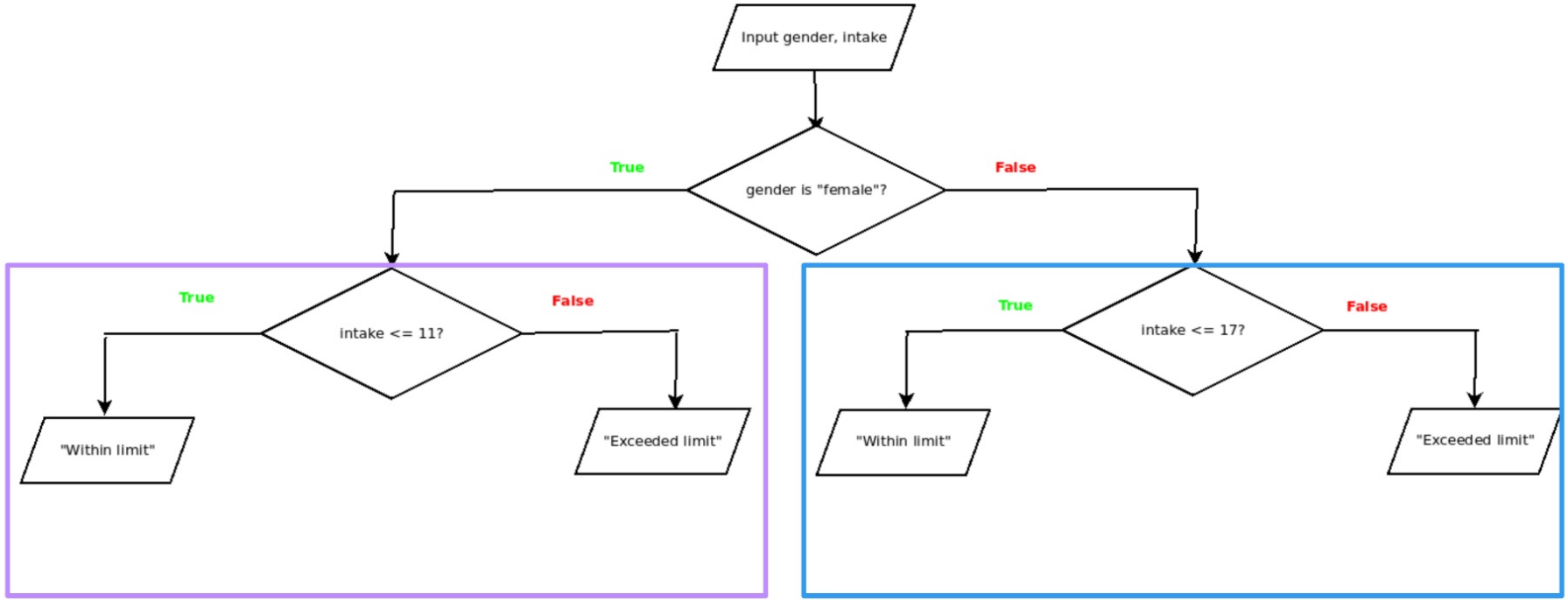
This means you can include a **if-elif-else** structure within an **if**, **elif** or **else**.

This is called a *Nested If*.

For example, the HSE specifies low-risk drinking guidelines for women and men:

Gender	Limit
Female	Up to 11 standard drinks in a week
Male	Up to 17 standard drinks in a week

A program to check if the number of units a person consumed exceeds the recommended limit could be implemented using the following logic:



Python Program

```
print("This program checks if you have exceeded the recommended weekly alcohol limit")

# Input gender
gender = input("Enter your gender (Male/Female): ")

# Input number of units consumed
units = float(input("Number of units of alcohol consumed this week: "))

# check if the user has exceeded her/his weekly limit
if gender.lower() == "female":
    if units > 11:
        print("You have exceeded the recommended alcohol limit for a woman")
    else:
        print("You have not exceeded the recommended alcohol limit for a woman")
elif gender.lower() == "male":
    if units > 17:
        print("You have exceeded the recommended alcohol limit for a man")
    else:
        print("You have not exceeded the recommended alcohol limit for a man")
else:
    print("Unable to process gender input")
```

Sample Output (Program Executed 4 Times)

```
Enter your gender (Male/Female): Female

Number of units of alcohol consumed this week: 15
You have exceeded the recommended alcohol limit for a woman


Enter your gender (Male/Female): Female

Number of units of alcohol consumed this week: 8
You have not exceeded the recommended alcohol limit for a woman
```

if within an **if**

```
if gender.lower() == "female":
    if units > 11:
        print("You have exceeded the recommended alcohol limit for a woman")
    else:
        print("You have not exceeded the recommended alcohol limit for a woman")
```

```
Enter your gender (Male/Female): Male

Number of units of alcohol consumed this week: 0
You have not exceeded the recommended alcohol limit for a man


Enter your gender (Male/Female): Male

Number of units of alcohol consumed this week: 24
You have exceeded the recommended alcohol limit for a man
```

if within an **elif**

```
elif gender.lower() == "male":
    if units > 17:
        print("You have exceeded the recommended alcohol limit for a man")
    else:
        print("You have not exceeded the recommended alcohol limit for a man")
```

Decision Making with and or not

You can implement more complex decision using the logical operators **and or not** to combine and/or negate (invert) conditions.

For example, amusement parks have age and height restrictions for their rides: children younger than 3, or less than 1.02 metres in height, may not use the ride.

```
print("This program checks if a child is permitted to use a ride")

# Input age
age = int(input("Enter child's age: "))

# input height
height = float(input("Enter child's height: "))

# check eligibility
if age < 3 or height < 1.02:
    print("Child is not permitted to use the ride")
else:
    print("Child is permitted to use the ride")
```

Sample Outputs

```
This program checks if a child is permitted to use a ride
Enter child's age: 2
Enter child's height: 1.1
Child is not permitted to use the ride
```

```
This program checks if a child is permitted to use a ride
Enter child's age: 2
Enter child's height: 0.9
Child is not permitted to use the ride
```

```
This program checks if a child is permitted to use a ride
Enter child's age: 4
Enter child's height: 0.95
Child is not permitted to use the ride
```

```
This program checks if a child is permitted to use a ride
Enter child's age: 3
Enter child's height: 1.1
Child is permitted to use the ride
```

Another way of implementing this program is using **and**

```
print("This program checks if a child is permitted to use a ride")

# Input age
age = int(input("Enter child's age: "))

# input height
height = float(input("Enter child's height: "))

# check eligibility
if age >= 3 and height >= 1.02:
    print("Child is permitted to use the ride")
else:
    print("Child is not permitted to use the ride")
```

A child is permitted to use the ride if s/he is 3 or older and at least 1.02m in height.

Notice how the less than signs **<** are replaced with greater than or equals **>=**

Example: Username Validation

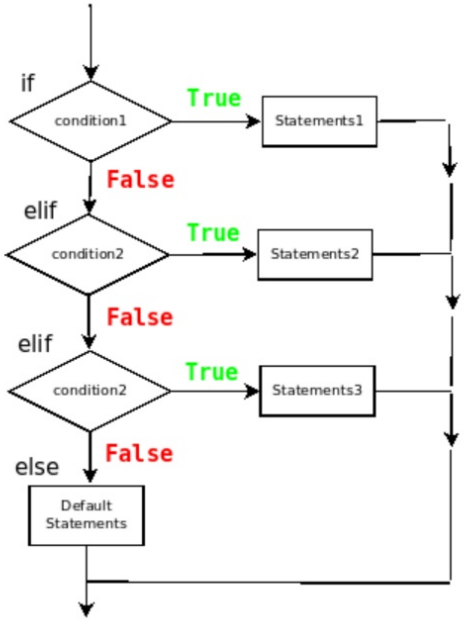
```
print("This program validates a username")

# Input username
username = input("Enter username: ")

# Check if the username is valid
if len(username) <= 15 and username.islower():
    print("Username is acceptable")
else:
    print("Username does not meet requi
```

Example	and		or	
Ride Restriction	age >= 3	height >= 1.02	age < 3	Height < 1.02
Username	len(username) <= 15	username.islower()	len(username) > 15	not username.islower()

```
print("This program validates a username")

# Input username
username = input("Enter username: ")

# Check if the username is valid
if len(username) > 15 or not username.islower():
    print("Username does not meet requirements")
se:
    print("Username is acceptable")
```

When **and** switches to **or**, and vice versa, the individual conditions are reversed:

Decision Making with **if-elif-else**



1

```
if condition:
    statements
```

2

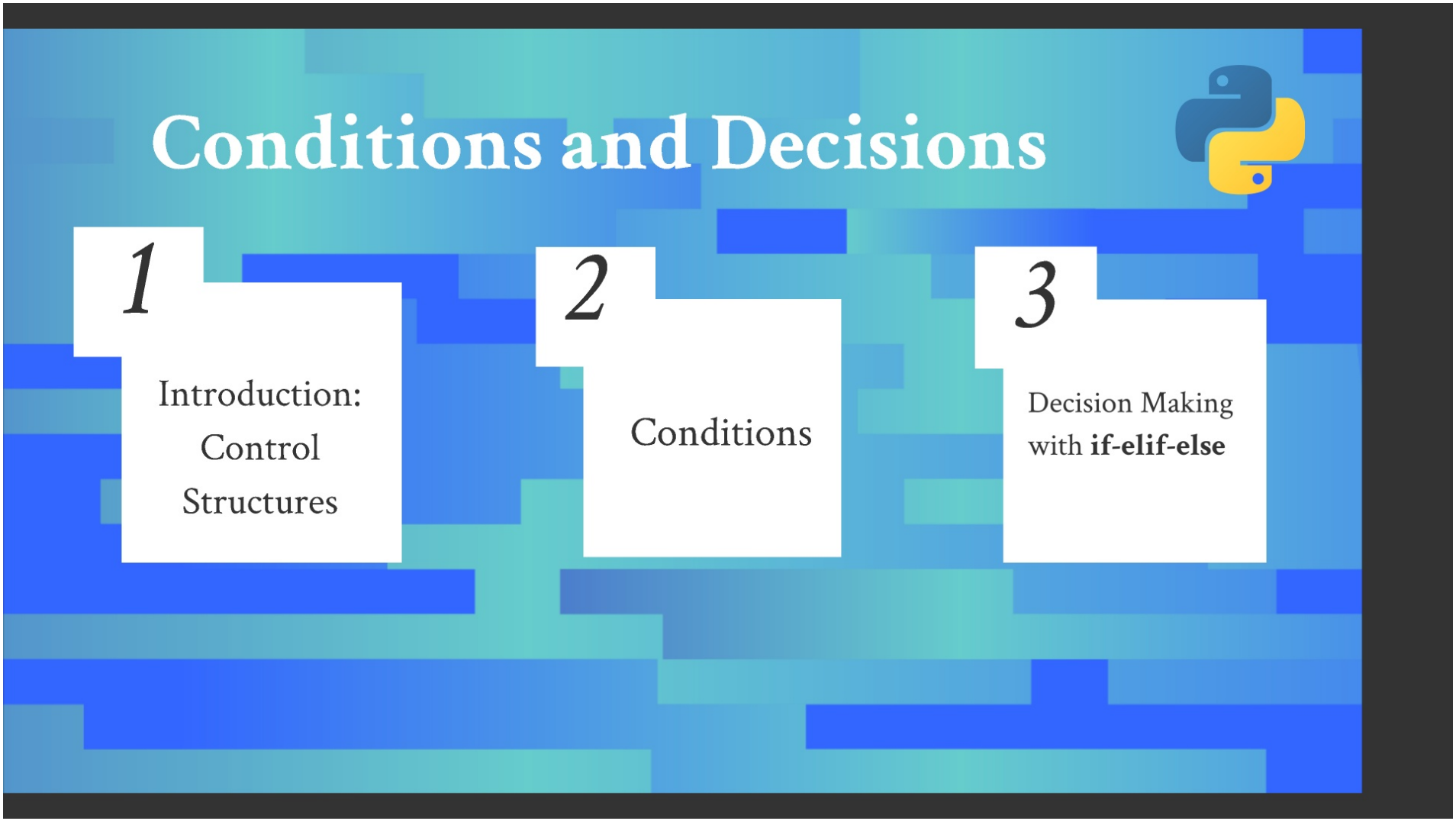
```
if condition:
    statements1
else:
    statements2
```

3

```
if condition1:
    statements1
elif condition2:
    statements2
else:
    statements3
```

4

Additional Topics


Conditions and Decisions
1
Introduction: Control Structures
2
Conditions
3
Decision Making with **if-elif-else**