Programming in Python

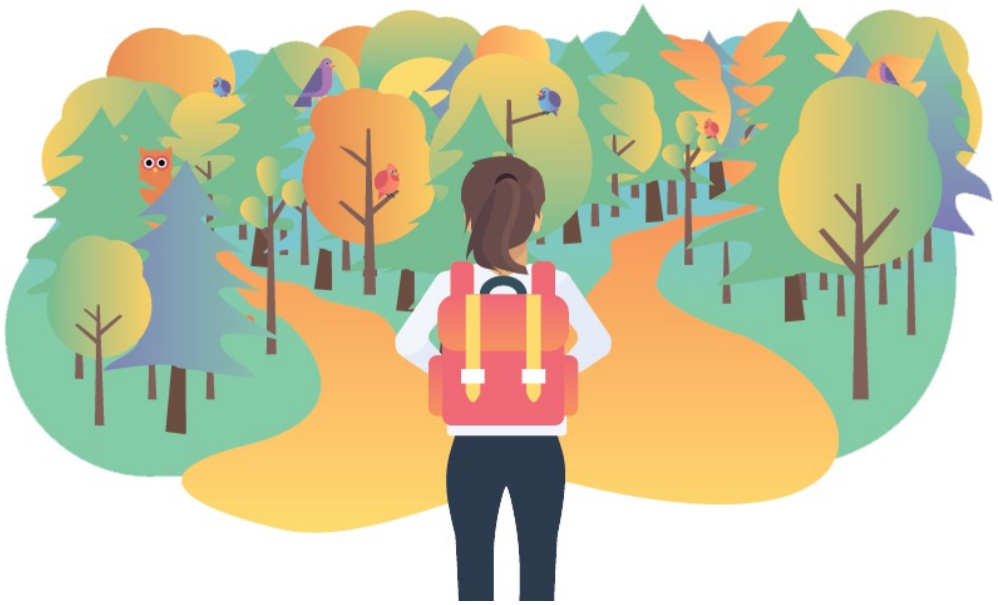
Branching

Flow of Control

- A program's flow of control is the order that the computer performs the statements in the code.
- Up to now, our programs have been sequential.
  - We start at the top and go line by line.
- What if we want to run certain lines only if some conditions are met?
- Think about when you enter a username and password.
  - You only get to proceed if you get them both correct!
- Branching statements let us choose between multiple options.
  - This is a fundamental part of computer programming.

Conditional Statements

- We want certain code to execute based on whether something is true.
  - If it is raining, then bring an umbrella.
  - If it is hot, wear shorts. If not, wear pants.
- In Python, we use:
  - **`if`**
  - **`if-else`**
  - **`if-elif-else`**



Using if

- An `if` statement allows us to determine whether some code will execute or not based on a condition.
- All `if` statements have a condition, which you can think of as a "Yes or No" question.
  - If the user's age is 18 or older, then they can vote.
- The condition must evaluate to be `True` or `False` (which is a `bool` value).
  - An expression that evaluates to `True` or `False` is called a **boolean expression**.
  - Operators that evaluate to `True` or `False` are called **boolean operators**.

Condition

- Syntax

```
if condition:
    statement1
```

```
if condition:
    statement1
else:
    statement2
```

- Place a colon `:` after the condition.
- You must indent the lines underneath the **`if`** statement.
- The **`else`** is optional.

Example

```
age = int(input("Enter your age: "))
if age >= 18:
   print("You can vote!")
else:
   print("Not yet")
```

```
Enter your age: 18
You can vote!
```

```
Enter your age: 17
Not yet
```

Indentation matters!

Comparison Operators

Operator	Meaning	Sample Condition	Evaluates To
`==`	equal to	`5 == 5`	`True`
`!=`	not equal to	`8 != 5`	`True`
`>`	greater than	`3 > 10`	`False`
`<`	less than	`5 < 8`	`True`
`>=`	greater than or equal to	`5 >= 10`	`False`
`<=`	less than or equal to	`5 <= 5`	`True`

Program

- Write a program that asks the user how many pets they have.
  - If the value is less than or equal to 2, tell them to adopt a pet.
  - Regardless of the value, print out a thank you message.
- Here is an example with user input.

```
How many pets do you have? 1
Please adopt a pet!
Thank you for using my program.
```

```
How many pets do you have? 3
Thank you for using my program.
```

Example

```
pets = int(input("How many pets do you have? "))
if pets < 2:
  print("Please adopt more!")
print("Thank you for using my program.")
```

```
How many pets do you have? 1
Please adopt a pet!
Thank you for using my program.
```

```
How many pets do you have? 3
Thank you for using my program.
```

Multiple Conditions

- Sometimes we may want different things to happen based on different conditions.
  - For example, if the score is greater than or equal to 90, the user gets an A, but if it is greater than or equal to 80, the user user gets a B, and so on.
- In Python, we can use **`if-elif-else`**

```
if condition:
    statement1
elif condition2:
    statement2
else:
    default
```

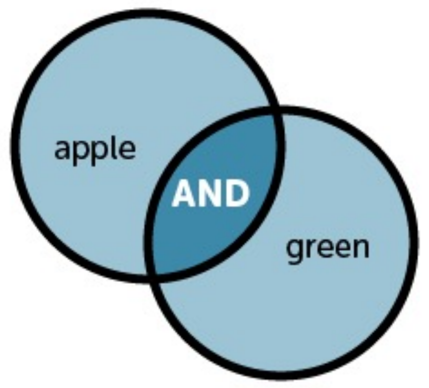
Example

```
score = int(input("Enter score: "))
if score >= 90:
   grade = "A"
elif score >= 80:
   grade = "B"
elif score >= 70:
   grade = "C"
elif score >= 60:
   grade = "D"
else:
   grade = "F"
print("Grade: " + grade)
```

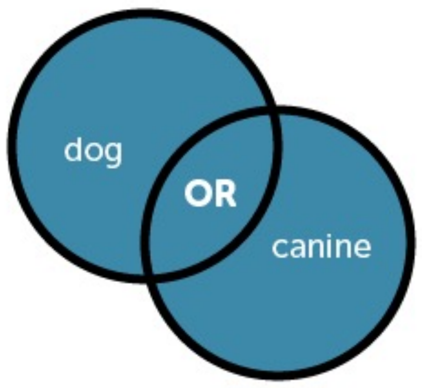
```
Enter score: 85
Grade: B
```

Boolean Logic

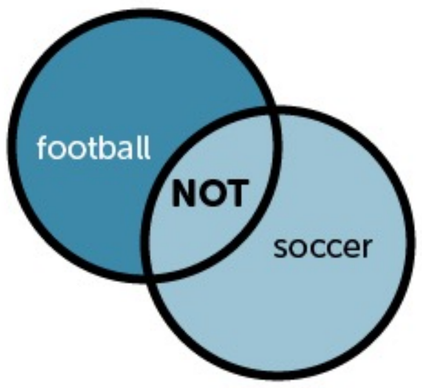
- Branching statements let us choose between multiple options.
- Sometimes we want multiple conditions to be true.
- Boolean logic allows us to use **`and`**, **`or`**, and **`not`** operators to make more sophisticated conditions.



You will get results that contain:
both **apple** and **green**



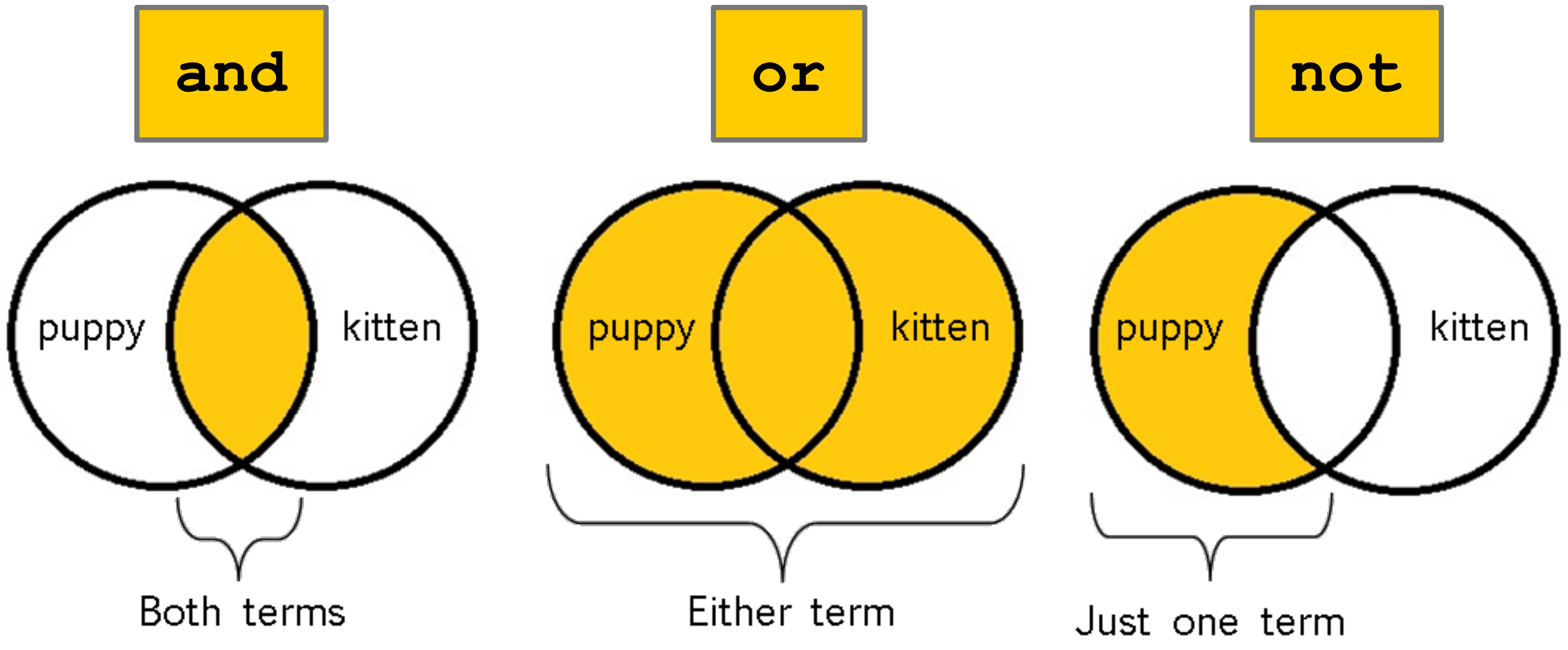
You will get results that contain:
either **dog**, or **canine**, or both



You will get results that contain:
football, and not **soccer**

Logical operators

- **`and`** & **`or`** combine two Boolean expressions



- **`not`** is used on one Boolean expression

`and` Operator

- Both expressions must be **`True`** for the whole expression to be **`True`**

`expression1` **`and`** *`expression2`*

```
num = int(input("Enter a number (0-9): "))
if num >= 0 and num < 10:
   print("You entered a single digit positive number.")
else:
   print("You did not enter a correct number.")
```

```
Enter a number: 15
You did not enter a correct number.
```

```
Enter a number: 8
You entered a single digit positive number.
```

`or` Operator

- Only one expression must be **`True`** for the whole expression to be **`True`**

```
expression1 or expression2
```

```python
num = int(input("Enter a number: "))
if num == 8 or num == 24:
    print("You entered Kobe's jersey number.")
```

```
Enter a number: 15
```

```
Enter a number: 8
You entered Kobe's jersey number.
```

`or` Operator

- What if both expressions are **`True`**?

```
num1 = int(input("Enter a number: "))
num2 = int(input("Enter another number: "))

if num1 == 8 or num2 == 8:
    print("You entered my favorite number.")
```

```
Enter a number: 8
Enter another number: 8
You entered my favorite number.
```

`not` Operator

- The **`not`** operator requires one Boolean expression
- The expression must be **`False`** for the whole expression to be **`True`**

```
not expression
```

```
num = int(input("Enter a number: "))

if not num >= 0:
    print("You entered a negative number.")
```

```
Enter a number: 8
```

```
Enter a number: -99
You entered a negative number.
```

True & False Tables

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

not	
True	False
False	True