Python Conditions and If statements

recal logical operations from lecture 1

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

In [ ]:
```
# If statement:
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

In [ ]:
```
# elif branch
# The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

In [ ]:
```
# else branch
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

In [ ]:
```
# Short Hand If ... Else
a = 2
b = 330
print("A is bigger") if a > b else print("B is bigger")
```

In [ ]:
```
# One line if else statement, with 3 conditions:
a = 330
b = 330
print("A is bigger") if a > b else print("A = B") if a == b else print("B is bigger")
```

logical operators

The `and` keyword is a logical operator, and is used to combine conditional statements:

In [ ]:
```
# Test if a is greater than b, AND if c is greater than a:
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

The `or` keyword is a logical operator, and is used to combine conditional statements:

In [ ]:
```
# Test if a is greater than b, OR if a is greater than c:
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

In [ ]:
```
# Nested If
# You can have if statements inside if statements, this is called nested if statements.
x = 41
if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Python Loops

Python has two primitive loop commands:

- `while` loops
- `for` loops

In [ ]:
```
# With the while loop we can execute a set of statements as long as a condition is true.
# Print i as long as i is less than 6:
i = 1
while i < 6:
  print(i)
  i += 1
```

The break Statement

With the `break` statement we can stop the loop even if the `while` condition is true:

In [ ]:
```
# Exit the loop when i is 3:
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
print('finally i is equal to:', i)
```

The continue Statement

With the `continue` statement we can stop the current iteration, and continue with the next:

In [ ]:
```
# Continue to the next iteration if i is 3:
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Python For Loops

A `for` loop is used for iterating over a sequence (that is either a `list` , a `tuple` , a `dictionary` , a `set` , or a `string`).

With the `for` loop we can execute a set of statements, once **for each item** in a `list` , `tuple` , `set` etc.

In [ ]:
```
# Print each fruit in a fruit list:
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Looping Through a String

Even `strings` are iterable objects, they contain a sequence of characters:

In [ ]:
```
# Loop through the letters in the word "banana":

for x in "banana":
  print(x)
```

In [ ]:
```
# break practice
```

In [ ]:
```
# continue practice
```

#The range() Function To loop through a set of code a specified number of times, we can use the `range()` function,

The `range()` function returns a sequence of numbers, starting from `0` by default, and increments by `1` (by default), and ends at a `specified number` .

In [ ]:
```
# Using the range() function:
for x in range(6):
  print(x)
```

In [ ]:
```
# Using the start parameter:
for x in range(2, 6):
  print(x)
```

In [ ]:
```
#Increment the sequence with 3 (default is 1):
for x in range(2, 30, 3):
  print(x)
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

In [ ]:
```
#Print each adjective for every fruit:
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
  for y in fruits:
    print(x, y)
```