COMP 204
While loop examples, nested while loops

Yue Li
based on materials from Mathieu Blanchette
Farenheit - Celsius conversion table

Goal: You are building a thermometer that needs to be graduated with both Celcius and Fahrenheit degrees. Write a program that computes and prints, for every temperature ranging from -40 °C to + 40°C, the corresponding temperature in Fahrenheit.

Expected output:
-40 °C = -40 °F
-39 °C = -38.2 °F
...
40 °C = 104 °F

General idea of algorithm:
- Use a while to iterate through all integers from -40 to +40
  - For each temperature, calculate Fahrenheit equivalent
  - Print result
Farenheit - Celsius conversion table

```python
tempCelsius = -40
while tempCelsius <= 40:
    tempFahrenheit = tempCelsius * 9 / 5 + 32
    print(tempCelsius, "C = ", tempFahrenheit, "F")
    tempCelsius = tempCelsius + 1
```
The guessing game

Write a program that implements the following game:

▶ First, the compute chooses a random integer between 1 and 10.
▶ Then the player has 5 guesses to find the number. For every guess, the program tells the player if it guessed too high or too low.
▶ The game ends when the player has guessed correctly, or when they used up their 5 attempts without success.

General idea of algorithm:

▶ Choose random number, save to variable
▶ Repeat the following, until 5 attempts are done or player made correct guess
  ▶ Ask for player’s guess
  ▶ Compare player’s guess to number, print appropriate message
The guessing game

```python
import random

hiddenNumber = random.randint(1,10) # Gives a random number between 1 and 10
correctGuess = False # Has player guess correctly yet?
nbGuesses = 0 # Keeps track of the number of guesses made

while correctGuess == False and nbGuesses < 5:
    guess = int(input("Guess a number:"))
    nbGuesses = nbGuesses + 1
    if guess == hiddenNumber:
        print("Correct!")
        correctGuess = True
    elif guess < hiddenNumber:
        print("Too low, guess again")
    else:
        print("Too high, guess again")

if correctGuess:
    print("You win!")
else:
    print("You lose!")
```
The break statement

Sometimes it is useful to stop executing the body of the loop mid-way through its execution, without waiting for the execution to return to the "while ... :" line.

```python
while booleanCondition:
    # some code block 1
    if (otherBooleanCondition):
        break
    #some code block 2

# rest of program
```

- Line 1: booleanCondition is evaluated. If True, jump to line 2. If False, exit loop and jump to line 9.
- Line 2: beginning of the body of the loop
- Line 4-5: If otherBooleanCondition is True, break out of loop, jump to line 9. Else continue
- Line 7: rest of the body of the loop
- After Line 7: Jump back to line 1
- Line 9: rest of the program (outside loop)
The guessing game revisited

```python
import random

hiddenNumber = random.randint(1, 10)  # Gives a random number between 1 and 10

correctGuess = False  # Has player guess correctly yet?
bGuesses = 0  # Keeps track of the number of guesses made

while correctGuess == False and nbGuesses < 5:
    guess = int(input("Guess a number:")
    nbGuesses = nbGuesses + 1
    if guess < 1 or guess > 10:
        print("Are you stupid?")
        break
    if guess == hiddenNumber:
        print("Correct!")
        correctGuess = True
    elif guess < hiddenNumber:
        print("Too low, guess again")
    else:
        print("Too high, guess again")

if correctGuess:
    print("You win!")
else:
    print("You lose!")
```
A palindrome is a word (or sentence) that reads the same in the forward and reverse direction. Example: kayak, racecar, ...
Write a program that checks is a given string is a palindrome or not.

One possible algorithm:

1. Compare the first character to the last.
2. If they don’t match, it’s not a palindrome; stop.
3. If they match, continue with the next position

... until all the first half of the word has been checked
word = input("Type a word: ")
wordLength = len(word)
index = 0  # used to scan the positions in the word
isPalindrome = True

while index < wordLength / 2:
    if word[index] != word[wordLength - index - 1]:
        # could also write if word[index] != word[ -(index+1) ]:
        isPalindrome = False
        break  # no need to continue looking at the rest,
        # so we break the loop
    index = index + 1  # don't forget this. Otherwise
    # you get an infinite loop

if isPalindrome:
    print("This is a palindrome")
else:
    print("This is not a palindrome")
A solid password should include at least one lowercase letter, one uppercase letter, one number, and one special character. Write a program that checks that a given password is solid. One possible algorithm:

- Ask user to type in password; save it in a string
- Count the number of lower, upper, number, special character (need counter variables for each)
  - for each position in the password string,
    - determine type of character
    - increase (increment) the corresponding counter variable
- check that all four counter variables are at least 1
While loops - password checking

```python
password = input("Type a password")

nbLowerCase = nbUpperCase = nbNumber = nbSpecial = 0
index = 0
while index < len(password):
    current = password[index]
    if current >= 'A' and current <= 'Z':
        nbUpperCase = nbUpperCase + 1
    elif current >= 'a' and current <= 'z':
        nbLowerCase = nbLowerCase + 1
    elif current >= '0' and current <= '9':
        nbNumber = nbNumber + 1
    else:
        nbSpecial = nbSpecial + 1
    index = index + 1

if nbLowerCase < 1:
    print("Must include a lowercase character")
if nbUpperCase < 1:
    print("Must include an uppercase character")
if nbNumber < 1:
    print("Must include a number")
if nbSpecial < 1:
    print("Must include a special character")
```
Nested loops

Just like nested conditionals, we can have nested loops.

```python
while booleanExpression1:
    # beginning of the outer loop
    while booleanExpression2:
        # body of the inner loop
        # rest of the outer loop
    # rest of program (outside while loop)
```

Execution:

- Line 1: booleanCondition1 is evaluated. If not true, jump to line 7. If true go to line 2
- Line 2: execute ”beginning of outer loop”
- Line 3: booleanCondition2 is evaluated. If not true, jump to line 5. If true go to line 4
- Line 4: Execute body of inner loop
- After line 4: Return to line 3
- Line 5: execute rest of outer loop
- After line 5: Return to line 1
- Line 7: execute rest of program
Nested loops - BMI table

Task: Print the BMI for every combination of weights and heights. Weight should range from 50 kg to 70 kg (in increment of 10). Height should range from 1.6 m to 1.8m, in increment of 0.1m. Output should look like this: BMI for 50 kg, 1.6m is 19.53...
BMI for 50 kg, 1.7m is 17.30...
BMI for 50 kg, 1.8m is 15.42...
BMI for 60 kg, 1.6m is 23.43...
...
BMI for 70 kg, 1.8m is 21.60...

Algorithm:

- Use a while loop to iterate through weights, starting at 30, up to 120.
  - Use a second while loop to iterate through heights, starting at 1.0, up to 2.0
  - Calculate BMI from current values of weight and height, print
 Nested loops - BMI table

```python
weight = 50
while weight <= 70:
    height = 1.6
    while height <= 1.81:  # why 1.81 and not 1.8?
        BMI = weight / (height ** 2)
        print("BMI for", weight, "kg", height, "m is", BMI)
        height = height + 0.1
    weight = weight + 10
```
A prime number is a number that is divisible only by 1 and itself. Task: Print all prime numbers up to a given limit.

Algorithm:

- Use a while loop to enumerate each candidate number, starting from 2 up to maximum number
  - Test each candidate by using a second while loop that enumerates every possible factor of the candidate prime, from 2 up to sqrt(candidate)
  - If never found a factor, then the number is prime. Print it.
import math
maxNumber = int(input(" Enter max. number to consider "))
candidatePrime = 2
while candidatePrime <= maxNumber:
    isPrime = True  # By default the number is prime
    candidateFactor = 2  # Test at all possible factors
    # of candidatePrime, starting with 2
    while candidateFactor <= math.sqrt(candidatePrime):
        # if the remainder of the integer division is zero,
        # then candidateFactor is a factor of candidatePrime
        if candidatePrime % candidateFactor == 0:
            isPrime = False
            break;  # break out of the inner loop, since
            # we’ve found a factor
    candidateFactor = candidateFactor + 1
    if isPrime:
        print(candidatePrime)
    candidatePrime = candidatePrime + 1