COMP 204

More loop examples, nested loops

Yue Li
based on materials from Mathieu Blanchette

Quiz 7 password

Example 1: Farenheit to Celsius conversion table

Goal: Your 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 + 40C, 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 loop to iterate through all integers from -40 to +40
  - For each temperature, calculate Fahrenheit equivalent
  - Print result

Farenheit to Celsius conversion table

```
# for-loop version
for tempCelcius in range(-40,41):
    tempFahrenheit = tempCelcius * 9 / 5 + 32
    print(tempCelcius," C = ",tempFahrenheit,"F")
```

```
# while-loop version
tempCelcius = -40
while tempCelcius <= 40:
    tempFahrenheit = tempCelcius * 9 / 5 + 32
    print(tempCelcius," C = ",tempFahrenheit,"F")
    tempCelcius = tempCelcius + 1
```

Example 2: The guessing game

Write a program that implements the following game:

- First, the computer 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 (correct code)

```
1  import random
2
3  hiddenNumber = random.randint(1,10) # Gives a random number
4                                      # between 1 and 10
5  correctGuess = False  # Has player guess correctly yet?
6  nbGuesses = 0  # Keeps track of the number of guesses made
7
8  while correctGuess == False and nbGuesses<5:
9      guess = int(input("Guess an integer between 1 and 10: ")
       )
10     nbGuesses = nbGuesses + 1
11     if guess == hiddenNumber:
12         print("Bingo!")
13         correctGuess = True
14     elif guess < hiddenNumber:
15         print("Too low, guess again")
16     else:
17         print("Too high, guess again")
18
19 if correctGuess:
20     print("You win!")
21 else:
22     print("You lose!")
```

Debugging exercise: two errors in this code to fix

```
1  import random
2
3  hiddenNumber = random.randint(1,10) # Gives a random number
       between 1 and 10
4  correctGuess = False
5  nbGuesses = 0
6
7  while correctGuess == False and nbGuesses<5:
8      guess = input("Guess an integer between 1 and 10: ")
9      nbGuesses = nbGuesses + 1
10     if guess = hiddenNumber:
11         print("Bingo!")
12         correctGuess = True
13     elif guess < hiddenNumber:
14         print("Too low, guess again")
15     else:
16         print("Too high, guess again")
17
18 if correctGuess:
19     print("You win!")
20 else:
21     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 ...:`" or "`for ...`" line.

```
1 while booleanCondition:
2     # some code block 1
3
4     if (otherBooleanCondition):
5         break
6
7     #some code block 2
8
9 # 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

```
1  import random
2
3  hiddenNumber = random.randint(1,10) # Gives a random number
4                                      # between 1 and 10
5  correctGuess = False  # Has player guess correctly yet?
6  nbGuesses = 0  # Keeps track of the number of guesses made
7
8  while correctGuess == False and nbGuesses<5:
9      guess=int(input("Guess an integer between 1 and 10: "))
10     nbGuesses = nbGuesses + 1
11     if guess < 1 or guess > 10:
12         print("Invalid input!")
13         break
14     if guess == hiddenNumber:
15         print("Bingo!")
16         correctGuess = True
17     elif guess < hiddenNumber:
18         print("Too low, guess again")
19     else:
20         print("Too high, guess again")
21
22 if correctGuess:
23     print("You win!")
24 else:
25     print("You lose!")
```

Example 3: Palindrome

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

Palindrome

```
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")
```

Example 4: Password checking

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

Example 4: Password checking

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

nbLowerCase = nbUpperCase = nbNumber = nbSpecial = 0

for index in range(0, 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

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.

```
1 while booleanExpression1:
2     # beginning of the outer loop
3     while booleanExpression2:
4         # body of the inner loop
5     # rest of the outer loop
6
7 # 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 example 1 - 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, 1.6 m is 19.53
BMI for 50 kg, 1.7 m is 17.30
BMI for 50 kg, 1.8 m is 15.42
BMI for 60 kg, 1.6 m is 23.43
...
BMI for 70 kg, 1.8m is 21.60
```

Algorithm:

- Use a loop to iterate through weights from 50 to 70 by 10
  - Use an inner loop to iterate through heights from 1.0 to 2.0
  - Calculate BMI from current values of weight and height, print

Nested loops - BMI table

```
weight = 50

while weight <= 70:
    height = 1.6 # reset height to 1.6 INSIDE the loop
    while height < 1.9:
        BMI = weight/(height**2)
        print("BMI for", weight," kg,", height," m is ",BMI)
        height = height + 0.1
    weight = weight + 10
```

Nested loops - BMI table

```
weight = 50

while weight <= 70:
    height = 1.6 # reset height to 1.6 INSIDE the loop
    while height < 1.9:
        BMI = weight/(height**2)
        print("BMI for", weight," kg,", height," m is ",BMI)
        height = height + 0.1
    weight = weight + 10
```

```
# What's wrong with this code?
weight = 50
height = 1.6 # reset height to 1.6 OUTSIDE of the loop
while weight <= 80:
    while height < 1.9:
        BMI = weight/(height**2)
        print("BMI for", weight," kg,", height," m is ",BMI)
        height = height + 0.1
    weight = weight + 10
```

Nested loops - BMI table

```
weight = 50

while weight <= 70:
    height = 1.6 # reset height to 1.6 INSIDE the loop
    while height < 1.9:
        BMI = weight/(height**2)
        print("BMI for", weight," kg,", height," m is ",BMI)
        height = height + 0.1
    weight = weight + 10
```

```
# What's wrong with this code?
weight = 50
height = 1.6 # reset height to 1.6 OUTSIDE of the loop
while weight <= 80:
    while height < 1.9:
        BMI = weight/(height**2)
        print("BMI for", weight," kg,", height," m is ",BMI)
        height = height + 0.1
    weight = weight + 10
```

```
import numpy as np # for floating-point range function
for weight in range(50,80,10): # for-loop
    height = 1.6 # reset height to 1.6 INSIDE the loop
    for height in np.arange(1.6,1.9,0.1): # for-loop
        BMI = weight/(height**2)
        print("BMI for", weight," kg,", height," m is ",BMI)
```

Nested loops example 2 - Prime numbers

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 loop to enumerate each candidate number, starting from 2 up to the given number
  - Test each candidate by using a second loop that enumerates every possible factor of the candidate prime, from 2 up to squared root of the candidate number
  - If never found a factor, then the number is prime. Print it.

Nested loops - Prime numbers

```
1  import math
2  maxNumber = int(input("Enter max. number to consider: "))
3
4  candidatePrime = 2
5  while candidatePrime <= maxNumber:
6
7      isPrime = True  # By default the number is prime
8      candidateFactor = 2  # Test at all possible factors
9                           # of candidatePrime,starting with 2
10     while candidateFactor <= math.sqrt(candidatePrime):
11         # if the remainder of the integer division is zero,
12         # then candidateFactor is a factor of candidatePrime
       ,
13         # so candidatePrime is not prime
14         if candidatePrime % candidateFactor == 0:
15             isPrime = False
16             break;  # break out of the inner loop, since
17                     # we've found a factor
18
19         candidateFactor = candidateFactor + 1
20
21     if isPrime:
22         print(candidatePrime)
23
24     candidatePrime = candidatePrime + 1
```

Nested loops - Prime numbers

```
1  # for-loop version
2  import numpy as np
3  maxNumber = int(input("Enter max. number to consider: "))
4
5  candidatePrime = 2
6
7  for candidatePrime in range(2, maxNumber+1):
8
9      isPrime = True  # By default the number is prime
10     candidateFactor = 2  # Test at all possible factors
11                          # of candidatePrime,starting with 2
12     for candidateFactor in np.arange(2, np.sqrt(
       candidatePrime)):
13
14         if candidatePrime % candidateFactor == 0:
15             isPrime = False
16             break;  # if not prime break out of the inner
       loop
17     if isPrime:
18         print(candidatePrime)
```