COMP 204

More loop examples, nested loops

Mathieu Blanchette

Quiz 6 password

While loops - input validity

Goal: Ask the user to enter their age. Keep asking until a valid number is entered.

```python
isValid = False
ageString = ""
while not isValid:
    ageString = input("Enter your age: ")

    if not ageString.isdecimal(): # isdecimal checks if a
                                  # string represents a
                                  # valid decimal number
        isValid = False
    else:
        ageFloat = float(ageString) #convert string to float
        isValid = ( ageFloat>=0 and ageFloat<200 )

    if not isValid:

        print("Invalid input: ",ageString,". Try again")

print("Input", ageString, "is a valid age")
```

While loops - input validity, part II

Goal: Modify program so that it stops asking after 5 attempts

```python
validity = False
n_attempts = 0  # this will serve as a counter
while validity==False and n_attempts<5:
    ageString = input("Enter your age: ")
    n_attempts=n_attempts + 1  # or just write n_attemps+=1

    if ageString.isdecimal()==True:
        ageFloat = float(ageString) #convert string to float
        validity = ( ageFloat>=0 and ageFloat<200 )

    if validity==False:  # same as "if not validity:"
        print("Invalid input: ",ageString,". Try again")

if validity==True:   # same as "if validity:"
    print("Input", ageString, "is a valid age")
else:
    print("Too many failed attempts!")
```

For loops

As we see, `while` loop allows us to repeat the execution of a block of code, as long as a certain condition hold.

Another type of loop is called **for loop**:

```
1 for someVariable in someList:
2     # body of the loop
3 
4 #rest of code
```

Execution:

- Line 1: `someVariable` gets the value of the next element in `someList`.
- If this is the first turn of the loop, the next element is the first element in the list. If there is no next element, jump to line 4, else execute body of loop (Line 2).
- Line 2: Body of loop
- After 2: Jump back to line 1.
- Line 4: rest of the program (outside loop)

Sidetrack: the `range` function

The `range` function is often used in combination with `for` loops.

- `range(stop)`: integer list from 0 up to `stop-1`
- `range(start, stop)`: integer list from `start` up to `stop-1`
- `range(start, stop, step)`: integer list from `start` up to `stop-1` but with increment set by `step`

```
range(5) # 0, 1, 2, 3, 4
range(3,7) # 3, 4, 5, 6   (note: 7 is not included)
range(3,9,2) # 3, 5, 7  (Start at 3, up to but not
    including 9, in increments of 2)
range(5,0,-1) # 5, 4, 3, 2, 1  (Start at 5, down to
     but excluding 0, in increments of -1)
```

For loops - countdown example (vs while-loop version)

countdown_forLoop.py:

```
# countdown program (for-loop version)
duration = int(input("Enter countdown duration: "))

for counter in range(duration, -1, -1): # fixed range
    print(counter)

print("Lift-off!")
```

countdown_whileLoop.py:

```
# countdown program (while-loop version)
duration = int(input("Enter countdown duration: "))

while duration >= 0 :
    print(duration)
    duration = duration - 1 # decrease value of counter

print("Lift-off!")
```

while loops vs for loops

You can always replace a for loop with a while loop, and vice-versa. But there are times where using one is much simpler than the other.

Use a **while loop** when:

- The number of iterations is not known ahead of time, but depends on the results of some computation, or on some user input.

Use a **for loop** when:

- We want to repeat a block of code for a *fixed* number of times
- OR
- We want to perform the same operation on each element of a sequence (see next lecture)

Sidetrack: how to access substring in a string

```
name = "Watson"

# we can access individual characters from a string by
# specifying the index (position) of the character you want

firstLetter = name[0]    # = "W". Note: number of positions
                         # starts at zero, not 1

secondLetter = name[1]   # = "a"

lastLetter = name[6]   # wrong! Causes exception because
                       # name doesn't contain a position 6

correctLastLetter = name[5]   # = "n".

numChar = len(name)   # = 6. number of characters in string

lastLetter = name[ len(name) - 1 ]   # = "n". This is a
                 # more general way to get the last letter
```

Sidetrack: how to access substring in a string

```

# we can extract several consecutive characters
firstHalf = name[0:3]  # = "Wat". This extracts characters
                       # at positions 0, 1, and 2

secondHalf = name[3:6] # = "son". =This extracts characters
                       # at positions 3, 4, and 5 = "son"

middle = name[2:4] # = "ts"

#we can operate from the end of the string by giving negative indices
lastLetter = name[-1] # "n"

penultimateLetter = name[-2] # "o"

reverseName = name[::-1] # "nostaW"

revAllButFirst = name[5:0:-1] # "nosta"
```

How to iterate over a string using loops

Task: change every occurrence of ‘T’ to ’U’ to convert a DNA sequence to an RNA sequence

Before we see the solution code, let’s step back and think about how shall we approach this problem by hand:

- Here is a DNA sequence: `ACTGAGCTAGCT`

Points to think about:

1. Where do we save the converted RNA sequence?
2. How do we access each letter in the DNA sequence?
3. How do we go to the next letter and then next letter and so on in the DNA sequence?
4. How do we change every T to a U but keep other letters the same?

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

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

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

For loops vs while loop: DNA Transcription

Task: Write a program that looks at a DNA sequence (String) and produce a second String that is the corresponding RNA sequence. This simply involves changing every 'T' to a 'U'.

```python
# for-loop version (better choice than while-loop):
dna=input("Enter a DNA sequence: ")
rna=""
for index in range(0,len(dna)): # iterate thru fixed range
    if dna[index] == "T":
        rna = rna + "U"
    else:
        rna = rna + dna[index]
print("The RNA sequence is:", rna)
```

```python
# while-loop version
dna=input("Enter a DNA sequence: ")
rna=""
index = 0
while index < len(dna):
    if dna[index] == "T":
        rna = rna + "U"
    else:
        rna = rna + dna[index]
    index = index + 1 # increment index
print("The RNA sequence is:", rna)
```

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

```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 integer between 1 and 10: "))
    nbGuesses = nbGuesses + 1
    if guess == hiddenNumber:
        print("Bingo!")
        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!")
```

Debugging exercise: fix errors in this code

```python
import random

hiddenNumber = random.randint(1,10) # Gives a random number between 1 and 10
correctGuess = False
nbGuesses = 0

while correctGuess == False and nbGuesses<5:
    guess = input("Guess an integer between 1 and 10: ")
    nbGuesses = nbGuesses + 1
    if guess = hiddenNumber:
        print("Bingo!")
        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 ...:`" 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: Stop loop on invalid input

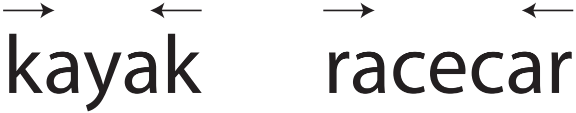
```python
import random

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 an integer between 1 and 10: "))
    nbGuesses = nbGuesses + 1
    if guess < 1 or guess > 10:
        print("Invalid input!")
        break
    if guess == hiddenNumber:
        print("Bingo!")
        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!")
```

Example 3: Palindrome

A palindrome is a word (or sentence) that reads the same in the forward and reverse direction. Example: kayak, racecar, ...
Task: 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



kayak racecar

Palindrome

```python
word = input("Type a word: ")
wordLength = len(word)
index = 0  # used to scan the positions in the word
isPalindrome = True

while index < wordLength/2:
    opposite_index = wordLength - index - 1
    if word[index] != word[ opposite_index ]:
    # 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

```python
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

```python
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
```

```python
# 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
#    for height in np.arange(1.6,1.9,0.1): # for-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

```python
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,
        # so candidatePrime is not prime
        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
```

Nested loops - Prime numbers

```python
# for-loop version
import numpy as np
maxNumber = int(input("Enter max. number to consider: "))

candidatePrime = 2

for candidatePrime in range(2, maxNumber+1):

    isPrime = True  # By default the number is prime
    candidateFactor = 2  # Test at all possible factors
                         # of candidatePrime,starting with 2
    for candidateFactor in np.arange(2, np.sqrt(candidatePrime)):

        if candidatePrime % candidateFactor == 0:
            isPrime = False
            break;  # if not prime break out of the inner loop
    if isPrime:
        print(candidatePrime)
```