COMP 204

Control flow - Loops

Yue Li
based on materials from Mathieu Blanchette

Quiz 6 password

Midterm time and location update

- Time: Friday, February 22 from 6:30-8:00 pm
- Location: LEA 219

Assignment #1 is posted on MyCourses on January 17

`https://www.cs.mcgill.ca/~yueli/teaching/COMP204_Winter2019/Assignments/HW1/assignment1.pdf`

Due date: February 1, 23:59

Submit one Python file per question, on MyCourses.

Start working on it ASAP!

Recap from last lecture: conditionals

- Conditionals (if-else, if-elif-else, and nested) allow us to decide which blocks of code get executed under which conditions.
- But each line of code is still executed either zero or one time.

```
1  if distance <= 20:
2      print("You must evacuate")
3  elif distance <= 40:
4      pregnant = input("Are you pregnant? (yes/no) ")
5      if (pregnant == "yes" or pregnant == "Yes" or pregnant
       == "Y" or pregnant == "y"):
6          print("You must evacuate")
7      else:
8          print("Evacuation is recommended")
9  else:
10     print("No need to evacuate")
```

Control flow: Loops

How do we execute the same operations multiple times?

Answer: **Loops**.

There are two types of loops:

1. `while` loop
2. `for` loop

```
1 while booleanExpression:
2     # body of the loop
3     # do something
4     # and some more
5
6 # rest of program (outside while loop)
```

What happens when this is executed?

- Line 1: booleanCondition is evaluated. If true, jump to line 2. If false, exit loop and jump to line 6.
- Line 2, 3, 4: the body of the loop is executed
- After line 4: Jump back to line 1
- Line 6: continue executing the rest of the program

The first loop example - countdown

```
1 # countdown program (while-loop version)
2 duration = int(input("Enter countdown duration: "))
3
4 while duration >= 0 :
5     print(duration)
6     duration = duration - 1 # decrease value of counter
7
8 print("Lift-off!")
```

Let's execute it step by step to see what happens ...

Input checking

In the examples seen so far, we did not do a very good job of check the validity of data entered by the user.

Usually, if a user enters invalid data, we should them ask to enter the data again.

General algorithm:

1. Ask user to enter some data (String)
2. Check the validity of the data
3. If the data is invalid, return to step (1), else continue with rest of program

While loops - input validity

```
1  isValid = False
2  ageString = ""
3  while not isValid:
4      ageString = input("Enter your age: ")
5
6      if not ageString.isdecimal(): # isdecimal checks if a
7                                    # string represents a
8                                    # valid decimal number
9          isValid = False
10     else:
11         ageFloat = float(ageString) #convert string to float
12         isValid = ( ageFloat>=0 and ageFloat<200 )
13
14     if not isValid:
15         print("Invalid input: \"",ageString,"\". Try again",
       sep="")
16
17 print("Input", ageString, "is a valid age")
```

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

- start: Starting number of the sequence
- stop: Generate numbers up to, but not including this number.
- step: Step size to increment the start until stop (default: 1).

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

In fact, 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
- We want to perform the same operation on each element of a object

For loops vs while loop example 2: password guessing

Task: keep asking password until get the correct one

```
# while-loop version (better choice than for-loop):
pwd=""
while pwd != "comp204":
    pwd=input("Enter the password: ")
print("Bingo! Good-bye!")
```

```
# for-loop verision
pwd=""
for i in range(2**32): # infinity
    pwd=input("Enter the password: ")
    if pwd == "comp204":
        print("Bingo! Good-bye!")
        break # break out of the loop
```

Resumed from the end of last lecture (01/18)

Sidetrack: how to access substring in a string

```
1  name = "Watson"
2
3  # we can access individual characters from a string by
4  # specifying the index (position) of the character you want
5
6  firstLetter = name[0]    # = "W". Note: number of positions
7                           # starts at zero, not 1
8
9  secondLetter = name[1]   # = "a"
10
11 lastLetter = name[6]   # wrong! Causes exception because
12                        # name doesn't contain a position 6
13
14 correctLastLetter = name[5]   # = "n".
15
16 numChar = len(name)   # = 6. number of characters in string
17
18 lastLetter = name[ len(name) - 1 ]   # = "n". This is a
19                  # more general way to get the last letter
```

Sidetrack: how to access substring in a string

```
1
2 # we can extract several consecutive characters
3 firstHalf = name[0:3]  # = "Wat". This extracts characters
4                        # at positions 0, 1, and 2
5
6 secondHalf = name[3:6] # = "son". =This extracts characters
7                        # at positions 3, 4, and 5 = "son"
8
9 middle = name[2:4] # = "ts"
10
11 #we can operate from the end of the string by giving
       negative indices
12 lastLetter = name[-1] # "n"
13
14 penultimateLetter = name[-2] # "o"
15
16 reverseName = name[::-1] # "nostaW"
17
18 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?

For loops vs while loop example 3

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

```
# 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)
```

```
# 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)
```

Quiz 6: finding start codon

Here are some RNA sequences:

AUGUGA :	start codon position at 0
ACCUAUGACGUCCUAAGCAGUUUGACG :	start codon position at 4
ACCUAUGAUAACCUAAGCAGUUUGACG :	start codon position at 4
AUGAUGAUGAUGAUGUGAUGAUGAUGA :	start codon position at 0
CGUAUUGCAGUGGUACUAC :	`contains no start codon`

Points to think about:

1. We want to go letter by letter meanwhile "look ahead" 3 letters ahead
2. We want to quit searching as soon as we find 'AUG'
3. We do not want to go outside of the range of the string sequence in case the string do not contain 'AUG'

Let's try out the 4 choices of while loop conditionals in Python