COMP 204
For loops, nested loops

Mathieu Blanchette, based on material from Christopher Cameron and Carlos Oliver Gonzalez
The for loop

We’ve seen the while loop before, which allows us to repeat the execution of a block of code, as long as a certain condition hold. Another type of loop is called a **for loop**:

```
for someVariable in someList:
    # body of the loop

# 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)
The for loop - example

```python
L = ['Andrea', 'Oliver', 'Chris', 'Malika']

for name in L:
    print("Dear", name, ", welcome to COMP 204.")
```

Prints:
Dear Andrea, welcome to COMP 204.
Dear Oliver, welcome to COMP 204.
Dear Chris, welcome to COMP 204.
Dear Malika, welcome to COMP 204.

Equivalent with while loop:

```python
L = ['Andrea', 'Oliver', 'Chris', 'Malika']

for name in L:
    print("Dear", name, ", welcome to COMP 204.")
```
For loops vs while 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 **for loop** when:

- You want to repeat a block of code for a *fixed* number of times
- OR You want to perform some operation on each member of a list

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.
The for loop - example

Goal: Calculate the average, minimum, and maximum of a list of grades.

grades = [68, 86, 77, 79, 73, 89, 56, 68]

sumGrades = 0 # will keep track of
# sum of grades seen so far
maxGrade = -1 # will keep track of max grade seen so far
# Why do we initialize to -1 ?
minGrade = 100 # will keep track of min grade seen so far
# Why do we initialize to 100 ?

for g in grades:
    sumGrades = sumGrades + g
    if g > maxGrade:
        maxGrade = g
    if g < minGrade:
        minGrade = g

print("Average: ", sumGrades/len(grades))
print("Minimum grade: ", minGrade)
print("Maximum grade: ", maxGrade)
What is an iterable?

For loops are used to *iterate over* a sequence, i.e. to visit each element of the sequence, and do some operation for each element.

In Python, we say that an object is an **iterable** when your program can *iterate over* it.

- Or an **iterable** is an object that represents a sequence of one or more values.

All instances of Python’s sequence types are iterables:

- Lists
- Strings
- Tuples
- More later...
The range function

The range function is often used in combination with for loops. It allows iterating through integers in a particular range:

- 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 excluding 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)

```python
duration = int(input("Enter countdown duration:"))
for counter in range(duration, -1, -1):
    print(counter)
print("Lift-off")
```
Iterating through a list of tuples

```python
periodicTable = [ ( "H" , 1 ) , ( "C" , 12) , ( "N" , 14) ]

# iterating through periodicTable with for loop
for element in periodicTable:
    print( element[0],"has mass",element[1],"g/mol" )
    print()

# Alternate, better way to do this:
for element,mass in periodicTable:
    print( element,"has mass",mass,"g/mol" )
    print()

# same thing with while loop
index = 0
while index < len(periodicTable):
    element = periodicTable[index]
    print( element[0], "has mass", element[1],"g/mol" )
    index = index + 1
```
Nested for loop

Just like nested conditionals, we can have nested loops.

```python
for someVariable1 in someList1:
    # start of body of outer loop

    for someVariable2 in someList2:
        # body of inner loop

        # rest of body of outer loop

    # rest of body of outer loop

# rest of program
```

- Line 1: Start of outer loop: someVariable1 gets the value of the next element in someList1. If there is no next element, jump to line 9, else execute body of loop (Line 2).
- Line 2: Start of body of outer loop
- Line 4: Start of inner loop: someVariable2 gets the value of the next element in someList2. If there is no next element, jump to line 7, else execute body of inner loop (Line 5).
- Line 5: Body of inner loop
- After Line 5: Jump back to line 4
- Line 7: rest of the body of outer loop
- After Line 7: Jump back to line 1
- Line 9: Rest of program (outside outer loop)
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 for loop to iterate through weights, starting at 50, up to 70.
  - Use a second for loop to iterate through heights, starting at 1.6, up to 1.8
  - Calculate BMI from current values of weight and height, print
```python
for weight in range(50, 71, 10):  # 50, 60, 70
    print("Starting to process weight:", weight)

    for heightInCm in range(160, 181, 10):  # 160, 170, 180
        heightInM = heightInCm / 100
        BMI = weight / (heightInM ** 2)
        print("BMI for", weight, "kg,", heightInM, "m is", BMI)

    print("Finished processing weight:", weight)

print("Done!")
```

Why do we iterate of heights in cm rather than m?

▶ Because the range function can only iterate over integers.
Nested while loops

We can also nest while loops.

```python
while boolean_expression1:
    # beginning of the outer loop
    while boolean_expression2:
        # body of the inner loop
        # rest of the outer 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

```python
weight = 50
while weight <= 70:
    heightInCm = 160
    while heightInCm <= 180:
        heightInM = heightInCm / 100
        BMI = weight / (heightInM ** 2)
        print("BMI for", weight, "kg,", heightInM, "m is", BMI)
        heightInCm = heightInCm + 10
    heightInCm = heightInCm + 10
weight = weight + 10
```
Nested for loops - Iterating through complex lists

Nested for loops are useful to iterate through lists where each item is itself a compound type:

```python
moleculesWithNames = [ ("carbon dioxyde", ["C","O","O"]),
                      ("nitrous oxyde", ["N","O"])]

for moleculeTuple in moleculesWithNames:
    print("Molecule name:", moleculeTuple[0])
    print("Molecule composition:")
    for atom in moleculeTuple[1]:
        print(atom)

# or an alternate, more elegant way:
for moleculeName, composition in moleculesWithNames:
    print("Molecule name:", moleculeName)
    print("Molecule composition:")
    for atom in composition:
        print(atom)
```
Often, it is useful to iterate through a sequence while keeping track of the index of the item we’re looking at. This can be done with the enumerate function.

```python
countriesSorted = [('China',140951739), ('India',1339180127), ('United States', 324459463), ('Indonesia',263991379)]

for rank, countryTuple in enumerate(countriesSorted):
    name = countryTuple[0]
    population = countryTuple[1]
    print(name, "has population", population)
    if rank == 0:
        print(name, "is the most populous country")
    if rank == len(countriesSorted) - 1:
        print(name, "is the least populous country")
```
Nested for loops: Diameter of a molecule

Goal: Given information about the 3D structure of a molecule (in the form of xyz coordinates for each of the atoms), find the diameter of the molecule, i.e. the distance between the two most distant atoms in the molecule.

Data: a list tuples, where each tuple contains a string describing the atom and a tuple containing the xyz coordinates of that atom.

Algorithm: Use nested for loops to over each pair of atoms in the molecule. For each firstAtom
  ▶ for each secondAtom
    ▶ Calculate distance between first and second atom
    ▶ If distance is larger than max distance seen so far, update max distance seen so far.
  ▶ Report max distance found
import math

aceticAcid = [ ("C", (0.053, 0.206, 0.000)),
    ("O", (0.053, 1.576, 0.000)),
    ("O", (1.274, -0.499, 0.000)),
    ("C", (-1.269, -0.557, 0.000)),
    ("H", (1.840, -0.238, 0.882)),
    ("H", (-1.840, -0.294, -0.890)),
    ("H", (-1.840, -0.294, 0.890)),
    ("H", (-1.070, -1.629, 0.000)) ]

maxDistance = -1

for firstAtom, firstXYZ in aceticAcid:
    for secondAtom, secondXYZ in aceticAcid:
        distance = math.sqrt((firstXYZ[0] - secondXYZ[0])**2 +
                              (firstXYZ[1] - secondXYZ[1])**2 +

        if distance > maxDistance:
            maxDistance = distance

print("Diameter of molecule is", maxDistance, "Angstrom")
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 for loop to enumerate each candidate number, starting from 2 up to maximum number
  - Test each candidate by using a second for 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 "))

for candidatePrime in range(2,maxNumber+1, 1):
    isPrime = True  # By default the number is prime

    # Test at all possible factors
    # of candidatePrime, starting with 2
    for candidateFactor in range(2, int(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

    if isPrime:
        print(candidatePrime)