Fundamentals of Programming 1

Lecture 7

Aurelia Power, ITB, 2018

The ‘while’ loop and selection

EXAMPLE: print only integers that are even, but also divisible by 3, from a given range.

- We need to start with the first even number of the range
- Increment by 2 (so we make sure that we only consider even numbers)
- We also need to repeatedly check whether the number is divisible by 3
- for instance, in the range 10 to 20, 12 and 18 are the only 2 numbers that satisfy both conditions.

PSEUDOCODE:

1. Create a variable counter to keep track of the number of repetitions and assign the value of the first even number of the range.
2. REPEAT WHILE/AS LONG AS the counter is less than or equal to the last value in the range:
   a) Check that the counter is divisible by 3:
      i. Output the value of the counter
   b) Update the value of the counter by 2

```
## prompt user to enter the beginning of the range
begin = int(input("Enter the starting integer: "))
## prompt user to enter the end of the range
end = int(input("Enter the end value: "))

## create a counter variable and give it the value of the first even integer
counter = begin
if begin % 2 != 0:
    counter = begin + 1

## as long as the counter is less or equal than the end value
while counter <= end:
    ## check that the counter is also divisible by 3
    if counter % 3 == 0:
        print(counter, end=' ')
    ## increase the counter by 2
    counter += 2
```

```
===== RESTART: C:/Users/Aurelia Powe
=====
Enter the starting integer: 10
Enter the end value: 20
12 18
>>>
```

The ‘while’ loop and selection

- Assume user entered **10** and **20**
- The program first assigns 10 to **counter** and checks to see whether it’s odd; it is even, so the statement **counter = begin + 1** will NOT be executed

Evaluation no.	while Boolean: counter <= end	Iteration	If Boolean: counter%3 == 0	counter += 2
1	10 <= 20 (True)	1	10 % 3== 0 (False)	counter = 10 + 2 (12)
2	12 <= 20 (True)	2	12 % 3== 0 (True) → output ’12’	counter = 12 + 2 (14)
3	14 <= 20 (True)	3	14 % 3== 0 (False)	counter = 14 + 2 (16)
4	16 <= 20 (True)	4	16 % 3== 0 (False)	counter = 16 + 2 (18)
5	18 <= 20 (True)	5	18 % 3== 0 (True) → output ’18’	counter = 18 + 2 (20)
6	20<=20 (True)	6	20 % 3== 0 (False)	counter = 20 + 2 (22)
7	20 <= 22 (False)	-	-	-

The ‘while’ loop and selection

EXAMPLE: prompt the user to repeatedly input words (finish the input taking when the user inputs ‘stop’) and output the number of those words that have 5 or more characters;

- We need a variable to keep track of the words of length 5 or more which initially is 0 (before the input taking)
- We need to prompt the user repeatedly to enter words until the user gets bored and finally enters ‘exit’
- We also need to repeatedly check whether each word enter has 5 or more characters.

PSEUDOCODE:

1. Create a variable **count** to keep track of the number of words that are 5 or more characters long.
2. Prompt the user to enter a word and assign it to a variable **word**
3. REPEAT WHILE/AS LONG AS **word** is not ‘stop’ :
   a) Check that the length of the word is >= than 5:
      i. Add 1 to the count
   b) prompt the user to enter another word or to stop
4. Output the count

```
## create a variable to keep track of words of 5 characters or more which 0 ini
count = 0
## prompt user for a word or 'stop' to finish
word = input("enter a word or 'stop' to exit: ")
## as long as the word is not 'stop'
while word != 'stop':
    ## check if the length of the word is >= 5
    if len(word) >= 5:
        ## add 1 to the count
        count += 1
    ## prompt user again for a word or 'stop' to finish
    word = input("enter a word or 'stop' to exit: ")
## output the value of the counter
print("the number of words that are 5 or more characters in length is", count)
```

```
========== RESTART: C:\Users\Aurelia Power\Desktop\2018_2019\POP\lect
enter a word or 'stop' to exit: blue
enter a word or 'stop' to exit: yellow
enter a word or 'stop' to exit: brown
enter a word or 'stop' to exit: black
enter a word or 'stop' to exit: red
enter a word or 'stop' to exit: stop
the number of words that are 5 or more characters in length is 3
>>>
```

The ‘while’ loop and selection

- Assume user entered **blue, yellow, brown, black, red, stop**
- Before entering the loop **count = 0** and **word = ‘blue’**
- Let’s track the values of each variable in the loop

Evaluation no.	while Boolean: word != ‘stop’	Iteration	If Boolean: len(word) >= 5
1	‘blue’ != stop (True)	1	len(‘blue’) >= 5 (False)
2	‘yellow’ != stop (True)	2	len(‘yellow’) >= 5 (True) → count = 0 + 1
3	‘brown’ != stop (True)	3	len(‘yellow’) >= 5 (True) → count = 1 + 1
4	‘black’ != stop (True)	4	len(‘yellow’) >= 5 (True) → count = 2 + 1
5	‘red’ != stop (True)	5	len(‘blue’) >= 5 (False)
6	‘stop’ != stop (False)	-	

The ‘while’ loop and selection

Your turn:

1. **Re-write the first example so that there is no selection in the while loop**

2. **Write a program that takes the FOP1 grades of all the students in year 1 (assume that there are 10 students for simplicity reasons, but the same logic applies to 100 or 200…) and determines how many students got As and how many students failed.**

The built-in range function

- Python provides a built-in function **range** that can be used to generate a sequence of integers that can be iterated over;
- **range(start, stop[, step])** where start specifies the beginning of the sequence, stop specifies where the sequence ends but the actual value is not inclusive, and step specifies the increment (or the decrement) value.

  Example: **range(2, 11, 2)** generates the sequence of integers 2, 4, 6, 8, 10; so it starts with 2, it increments by 2, and stops when reaches 11 but it does not include 11.

- We can leave out the start and step values in which case they will take the default values of 0 and 1, respectively.

  Example: **range(10)** generates the sequence of integers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 where the start has the default value of 0, the stop is 10 (not included in the actual sequence) and the step is the default 1 which means that it increments by 1.

The built-in range function

- We can leave out only the step value in which case it will take the default value of 1.

Example: **range(1, 11)** generates the sequence of integers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 where the start value is 1, the stop value is 11 (not included in the actual sequence) and the step is the default 1 which means that it increments by 1.

- We can also generate a backwards sequence.

Example: **range (9, 2, -3)** generates the sequence of integers 9, 6, 3, where the start value is 9, the stop value is 2 (not included in the actual sequence) and the step is -3 which means that it decrements by 3.

- **Exercise**: what are the sequences generated?

```
range(10, -1, -2); range(10, 101, 10);

n = 7; m = 50; x = 7; range(n, m, x)
```

The ‘for’ loop syntax

- A **for** statement/loop is an iterative statement that iterates once for each element in a specified sequence of elements.
- So, **for** is typically used to write a definite loop (to implement counter-controlled repetition);
- In other words, the number of repetitions is known before the execution of the loop;

k is the loop variable (we can name it anything as long as it is a valid identifier); this variable is updated in each iteration.

some_sequence is the sequence that contains the values that the loop variable (k) will take in each iteration. It could be a range, or a list, or a string, etc.; for now we will use only ranges.

```
for k in some_sequence:

    ## instruction(s) to be repeated go here
```

‘while’ vs ‘for’ – displaying the numbers 1 to 10

```
int counter = 1; ## Initialize the counter
while counter <= 10: ## Check counter is less or equal to 10
   print(counter, end = ' '); ## output the counter
   counter+=1; ## Update the counter
```

The above **while** loop can be written as a **for** loop in a more compact way.

```
for counter in range(1, 11):
   print(counter, end = ' ')
```

‘for’ – displaying the numbers 1 to 10

The loop variable **counter** will take a different value in each iteration; that value is decided by what it is contained in the actual sequence.

The sequence is that generated by the range function: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (11 is not included)

```
for counter in range(1, 11):
    print(counter, end = ' ')
```

Instruction that is repeated in each iteration: displaying (printing) the value of the **counter**

for loop - Displaying the numbers 1 - 10

- The sequence generated by the range function is: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- The **for** loop works in this case by iterating over all the numbers in the sequence in the order they appear.

- So, in the first iteration, the pointer points to the first value in the sequence (1), thus the value of counter is 1
- In the second iteration the pointer points to the second value in the sequence (2), so the value of counter is 2
- In the third iteration the pointer points to the third value in the sequence (3), so the value of counter is 3
- In the fourth iteration the pointer points to the second value in the sequence (2), so the value of counter is 2
- And so on, until there are no more values in the sequence.

‘for’ – displaying even numbers within a given range

```
- Display the numbers: 0, 2, 3, 4, 6, 8, 10
for counter in range(0, 11, 2):
   print(counter, end = ‘ ’)
```

```
- Display the numbers: 10, 12, 14, 16
for counter in range(10, 17, 2):
   print(counter, end = ‘ ’)
```

```
- Display the numbers: 0, -2, -4, -6, -8, -10
for counter in range(0,-11, -2):
   print(counter, end = ‘ ’)
```

‘for’ – other examples

```
- Display the cumulative sums of integers 0 - n;
- assume n already initialised
total = 0
for counter in range(0, n + 1):
    total += counter
    print(total, end = ‘ ’)
```

```
- Display the squared values of integers between n and m
- assume n and m already initialised
for i in range(n, m + 1):
    print(i**2, end = ‘ ’)
```

‘for’ – exercises

1. Re-write the problem of calculating the average mark for the fop1 mcq test applied to a group of 16 students (from the previous lecture).

2. How many numbers does this loop print?

```
for k in range(0, -10, 2):
    print(k)
```

3. What does the following block of code outputs?

```
s = 0;
for n in range(10):
    if n < 0:
        s = s + n
```

Break and continue statements

break and **continue** statements are used to alter the normal flow of a loop.

- The break statement terminates prematurely the loop that contains it.
- Let's consider the following **example**: prompt the user to repeatedly input words (finish the input taking when the user inputs '-1') and output the number of words entered;
- The way we can solve it is to use the sentinel '-1' in the while header;
- But we can achieve the same using selection and break statements: that is, as soon as the user enters '-1' terminate/break out of the loop.

```
count = 0
while(True):
        word = input("enter a word or '-1' to exit: ")
        if word == '-1':
                break
        count += 1
print("the number of words you entered is:", count)
```

Break and continue statements

- The continue statement skips the rest of the code in the loop that contains it only for the current.
- Let's consider the following **example**: output all the numbers between 10 and 20, but skip '13' because is unlucky (☺);
- The way we can solve it is to use selection and continue statements: if the number is 13, skip the code after it (in this case the print instruction) and continue with the rest of the iterations

```
for n in range(10, 21):
        if n == 13:
                continue
        print(n, end=' ')
```

```
10 11 12 14 15 16 17 18 19 20
>>>
```

Break and continue statements … your turn

```
for j in range(0, 5):
    print("iteration number: ", j)
    if(j == 3)
        break;
    print(j + 10)

for j in range(0, 5):
    print("iteration number: ", j)
    if(j == 3)
        continue;
    print(j + 10)
```

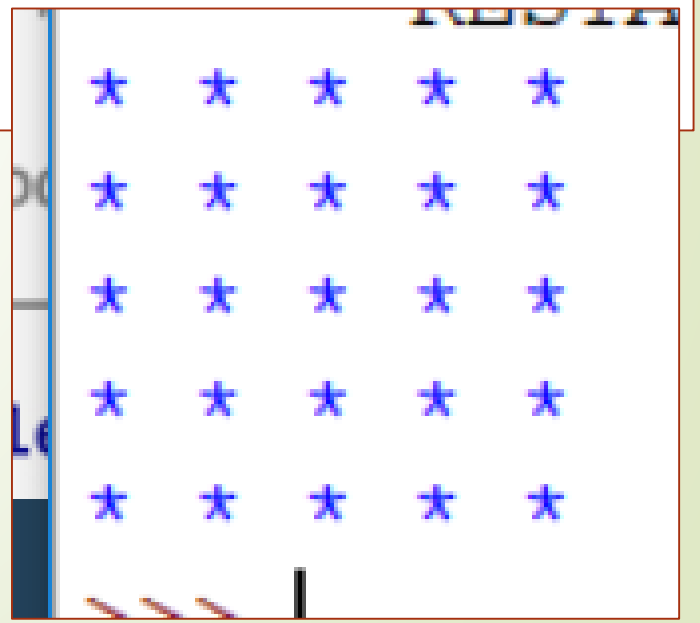
```
i = 0;
while i <= 3:
    print('hello number', i)
    if(i == 2):
        break
    i += 1

i = 0;
while i <= 3:
    print('hello number', i)
    if(i == 2):
        continue
    i += 1
```

Nested loops

- When the body of a loop contains another loop, the loops are **nested**.
- For instance, we want to display a square of 5 by 5 stars:

- Loop as many rows of stars, in this case 5
  - Inside each row, loop as many stars in a row, in this case also 5, and print 5 stars in a row
  - Move the cursor to a new line (to start a new row)

```
for i in range (0, 5):
        for j in range (0, 5):
                print('*', end=' ')
        print()
```



Nested loops – another example

- A typical use of nested loops is printing a table with rows and columns:
  - The outer loop iterates over all the rows
  - The inner loop iterates over all columns in the current row.
- For instance, we can display in tabular form all the powers of x from1 to 3, where x ranges from 1 to 3 also

```
## print the header of the table first
for i in range(1, 4):
        print("x **", i, '\t', end='')
## print an empty line so the cursor will be moved to a new line
print()
## print the actual values from the table
for i in range(1, 4): ## the outer loop select the value of x
        ## create a variable to refer to the value of exponentiated value
        x_updated = 1
        ## the inner loop multiplies the value of x into x_updated
        ## exactly 3 times, but if the end of the range was 10, then it
        ## would exactly 9 times...
        for j in range(1, 4):
                x_updated = x_updated * i
                print(x_updated, '\t', end='')
        ## print an empty line so the cursor will be moved to a new line
        print()
```

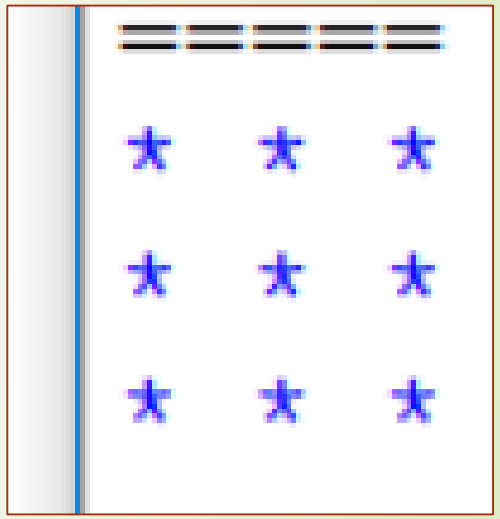
```
x ** 1   x ** 2   x ** 3
1        1        1
2        4        8
3        9        27
```

Nested loops – combining types of loops

- You can have any combination of nested loops
- Below there are 2 examples that do exactly the same thing

```
for n in range(0, 3): ## the outer loop is a for loop
        m = 0
        while m < 3: ## the inner loop is a while loop
                print("*", end=' ')
                m += 1
        print()
```

```
m = 0
while m < 3:## the outer loop is a while loop
        ## the inner loop is a for loop
        for n in range(0, 3):
                print("*", end=' ')
        m += 1
        print()
```



Nested loops – your turn

What is the output of the following code fragment?

```
m = 3
while m >= 1:
    for i in range(0, 2):
        print('a', end=' ')
    print()
    m -= 1
```

What is the output of the following code fragment?

```
for i in range(4):
    for j in range(4):
        if j % 2 == 0:
            print("x", end=' ')
        else:
            print("o", end=' ')
    print()
```

Generating random integers

- We will use the **random module** from the python 3 standard library which needs to be imported
- From this module we will use the method **randint(a, b)** which returns a random integer from that range, both ends inclusive:
- For example, random.randint(1, 3) could return 1, or 2 or 3
- Let's write a script to generate 5 random integers between 0 and 4 and run it several times

```
## import the module random
import random
for i in range(0, 5):
        ## use the function randint to generate
        ## a random integer between 0 and 4
        print(random.randint(0, 4), end=' ')
```

```
===== RESTAR
1 3 0 2 2
>>>
===== RESTAR
1 3 0 0 1
>>>
===== RESTAR
0 3 2 1 0
>>>
===== RESTAR
2 3 0 4 4
>>>
```

- Exercise: write a while loop to generate 10 random integers between -10 and 10; run that several times