Lecture 6: Iteration

COMP90059 Introduction to Programming

Wally Smith

School of Computing & Information Systems

ASSESSMENT - REVISED on 9th April 2020

% contributions to overall grade for the subject are shown for each piece of assessment

•**Assignment 1** (10%) is ongoing and due on Friday 24th April.

•**Mid-Semester Test** (0%) will take place on Week 8, at 2 - 3 pm on Monday 4th May. This will be a one-hour test online via Grok in place of the usual Monday lecture. Note that your score on the test will NOT contribute to your overall grade for the subject, but it will provide you with feedback on your learning so far, and it will also give you a good indication of what the final examination will be like.

•**Assignment 2** (20%) is a second set of programming tasks on Grok. It will be released on Wednesday 6th May (during Week 8, after the Mid- Semester Test) , and is due on Friday 29th May (Week 11).

•**Final Examination** (70%) will be held in the examination period after semester (date to be advised). It will be delivered online in a way to be advised. You will carry out the exam during a specified 3 hours.

•**Hurdle requirement: 35/70 or greater** on the Final Examination is needed to pass the subject.

Please rest assured that the Final Examination is not intended to present fiendishly complex problems. Instead, it will present a mixture of questions that are designed to allow you demonstrate the knowledge and skills you have developed through the lectures, tutes and assignments. The nature of the Final Examination will be discussed in lectures in the remainder of semester, and we will look at many examples of the kinds of questions that will be included.

Lecture Overview

Formatting strings and numbers

- Review the use of the f-string technique to format our output

Looking more closely at iteration

*Iteration is the technique of repeating lines of code many times within a program to get a desired outcome. We have already encountered **for loops** to do this, and in this lecture we will explore the technique further, ever mindful of the need to write elegant code.*

- **for loops** - revisiting them and working through some new applications
- **while loops** - learning this new technique, and considering when it is useful

Formatting outputs

string & number formatting

```
principal = eval(input('Enter the initial principal: '))
apr = eval(input('Enter the annual interest rate: '))
numberYears = int(input('Enter the number of years: '))
apr = apr/100
for i in range(numberYears):
        principal = principal * (1 + apr)
print('The final value is: ', principal)
```

```
Enter the initial principal: 1000
Enter the annual interest rate: 5.6
Enter the number of years: 6
The final value is:  1386.7031726716357
```

string & number formatting

Python supports different ways of formatting strings output - the most recent technique (which you should use) is called f-strings.

```
name = 'Jane'        age = 26        distance = 17.2300456
```

Embedding variables and expressions in the string literal to be printed

```
print(f'{name} is {age} years old and lives {distance} kms from the city centre')
                    Jane is 26 years old and lives 17.2300456 kms from the city centre
print(f'{name} is {age} years old and travels {distance *  2} kms each day')
                    Jane is 26 years old and travels 34.4600912 kms each day
```

Adding column width and justification

```
                                                                 20
print(f'Distance of {distance*2 :20}')         Distance of           34.4600912
print(f'Distance of {distance*2 :<20}')        Distance of 34.4600912
print(f'Distance of {distance*2 :^20}')        Distance of      34.4600912
```

Adding precision

```
print(f'Distance of {distance:20.8}')          Distance of            17.230046
print(f'Distance of {distance:20.2}')          Distance of              1.7e+01
print(f'Distance of {distance:20.2f}')         Distance of                17.23

print(f'Distance of {distance:^20.2f}')        Distance of        17.23
```

Exercise 1. Formatting the principal output

Change the last line of code using an f-string to get an output with 2 decimal places - ie to show dollars and cents (without the sign for now)

```
principal = eval(input('Enter the initial principal: '))
apr = eval(input('Enter the annual interest rate: '))
numberYears = int(input('Enter the number of years: '))
apr = apr/100
for i in range(numberYears):
        principal = principal * (1 + apr)
print('The final value is: ', principal)
```

```
Enter the initial principal: 1000
Enter the annual interest rate: 5.6
Enter the number of years: 6
The final value is:      1386.70
```

print() - changing the end parameter

print() statements write a series of expressions to the screen + a default return character

```
print('hello', 'how')
print('are', 'you')
print()
print('today')
```

```
hello how
are you

today
```

The default return character can be changed to another character ...

```
print('hello', 'how', end='')
print('are', 'you', end='')
print('today')
```

```
hello how are you today
```

```
print('why not', end='?')
```

```
why not?
```

Iteration

L4 The anatomy of programs (so far)

We can think of programs as containing the following:

- **data structures** of different types**,**

  e.g., integers, floating point numbers, Booleans,
  sequences (strings, lists)

- **functions** - actions to be carried out on data

  - e.g., print, input

- **control flow** - redirecting the line-by-line flow of code

  conditionals (if-elif-else)

  iteration (for loops, while loops)

Iteration

Repeating the same process many times to achieve a desired outcome.

In programming, iteration means creating a block of code that is repeated cyclically in a loop.

There are three main forms of iteration ...

- eternal - repeat something forever

  (e.g., Windows)
- definite or counted - repeat for a fixed number of times

  (e.g., move Mario forward 10 pixels; print 7 copies)
- indefinite or conditional - repeat while some condition remains true

  (e.g., scroll while button pressed)

L4 for loops

for loops are determinate or counted loops in which a code block is repeated a specified number of times.

```
for <variable> in <iterable>:
        <statement>
        <statement> ...
```

e.g.

```
for number in range(7):
        print(number)
```

```
0
1
2
3
4
5
6
```

```
for ch in 'Hello, how are you?':
        print(ch)
```

```
H
e
l
l
o
,
 
h
o
w
 
a
r
e
 
y
o
u
?
```

Exercise 2: for loops with integers

Write a program using a for loop to print out the 17 times table up to 1000.

ie...

```
1 x 17 = 17
2 x 17 = 34
3 x 17 = 51
...

1000 x 17 = 17000
```

Exercise 3: for loops with strings

Write a program using a for ... loop to count how many vowels are in the string:

'Do not pass go. Do not collect 200 dollars'

Clue 1: use the string as an iterable:

```
for ch in target:
```

Clue 2: you can test if a character is a vowel by:

```
if ch in 'aeiou'
```

L4 : nested for loops

A. Complete the program below so that it prints out the names of each month of the year in a continuous list down the page.

B. Modify your program below so that it prints out the letters of all the month names in one continuous stream.

```
months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
'September', 'October', 'November', 'December']
```

```
for month in months:
    print(month)
```

```
for month in months:
    for letter in month:
        print(letter)
```

```
January
February
March
April
May
June
July
August
September
October
November
December
```

```
J
a
n
u
a
r
y
F
e
b
r
u
a
r
y
M
a
r
c
h
A
p
r
i
l
M
a
y
J
u
n
e
J
u
l
y
```

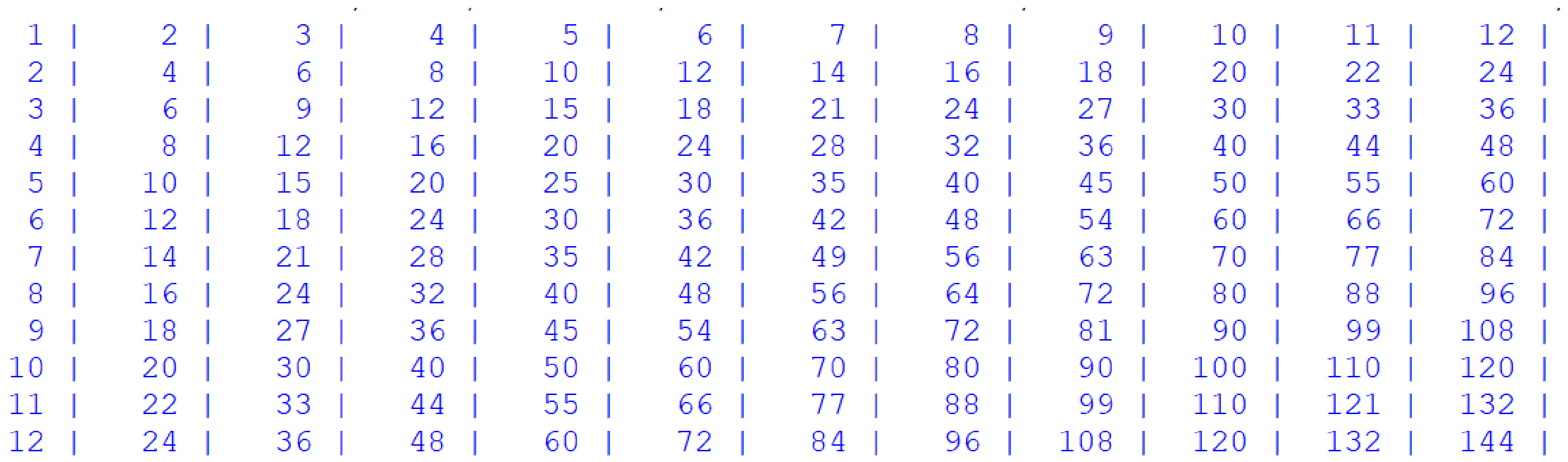
The syntax of nested for loops

```
for var1 in sequence1:
        statement(s)    #outer loop
        for var2 in sequence2:
              statements(s)    # inner loop
        statement(s)    # outer loop
```

Exercise 3: times tables

Write a program that produces the below chart of the times tables up to 12.

Use a nested for loop, f-string formatting & use the character ' | ' to make the lines.



```
    1 |     2 |     3 |     4 |     5 |     6 |     7 |     8 |     9 |    10 |    11 |    12 |
    2 |     4 |     6 |     8 |    10 |    12 |    14 |    16 |    18 |    20 |    22 |    24 |
    3 |     6 |     9 |    12 |    15 |    18 |    21 |    24 |    27 |    30 |    33 |    36 |
    4 |     8 |    12 |    16 |    20 |    24 |    28 |    32 |    36 |    40 |    44 |    48 |
    5 |    10 |    15 |    20 |    25 |    30 |    35 |    40 |    45 |    50 |    55 |    60 |
    6 |    12 |    18 |    24 |    30 |    36 |    42 |    48 |    54 |    60 |    66 |    72 |
    7 |    14 |    21 |    28 |    35 |    42 |    49 |    56 |    63 |    70 |    77 |    84 |
    8 |    16 |    24 |    32 |    40 |    48 |    56 |    64 |    72 |    80 |    88 |    96 |
    9 |    18 |    27 |    36 |    45 |    54 |    63 |    72 |    81 |    90 |    99 |   108 |
   10 |    20 |    30 |    40 |    50 |    60 |    70 |    80 |    90 |   100 |   110 |   120 |
   11 |    22 |    33 |    44 |    55 |    66 |    77 |    88 |    99 |   110 |   121 |   132 |
   12 |    24 |    36 |    48 |    60 |    72 |    84 |    96 |   108 |   120 |   132 |   144 |
```

Exercise 4: prime number generator

Write a program using a for ... loop to list all the prime numbers between 1 and 1000.

Recall that a prime number is one that has exactly two distinct factors: 1 and itself (e.g., 5, 7, 17 - but not 1)

Clue 1 : to tell if a number (x) is perfectly divisible by another number(y):

```
if x % y == 0
```

Clue 2 : use a Boolean variable to keep track of whether each value is a prime or not

while loops

A code block is repeated for as long a some condition is true.

```
while <Boolean variable>:
        <statement>
        <statement> ...
```

e.g.

```
i = 0
while i  < 10:
    print(i)
    i  += 1
```

```
0
1
2
3
4
5
6
7
8
9
```

Also known as indefinite or conditional loops.

Exercise 5: while loop

Write a program that takes string called 'target' and prints every letter until the letter e is detected.

Start with the assignment:

target = 'What is the time of the meeting?'

Exercise 6: duplicates

Write a program that takes a list of words and identifies if a word is duplicated in the list.

Exercise 7b: duplicates

Write another version of the duplicate_in function that uses for loops to achieve the same result of detecting a duplicate (True or False) by comparing every word in the list directly with every other word.

Choosing between **for** and **while** loops

for loops

- Given a choice between the two, for loops are generally more elegant/safer/easier to understand
- Use a for loop, if you need to iterate over all items of an iterable,

while loops

- Consider a while loop, if there is a well defined end-point to the iteration which doesn’t involve iterating over all items,
- While loops become powerful when unpredictable results or events arise during the iteration (often involving user input)

Using a while loop to improve the efficiency of the prime number generator code

```
# program to print primes numbers between 0, 1000 with nested for loops
for num in range(2, 1001):
    prime = True
    for divider in range(2, num):
        if num % divider == 0:
            prime = False
    if  prime:
        print(num)

# program to print primes numbers between 0, 1000 with for + while loops
for num in range(2, 1001):
    prime = True
    divider = 2
    while prime and divider < num:
        if num % divider == 0:
            prime = False
        divider += 1
    if prime:
        print(num)
```

Using a while loop with user input

Suppose that we want the user to enter a set of numbers for analysis, and we want them to decide how many items they give us.

```
# program to take data from the user and calculate mean

count = 0
total = 0
moredata = 'yes'

while moredata[0] == 'y':
   number = float(input('Enter a number: '))
   total += number
   count += 1
   moredata = input('Do you have more data (Y/N)')

print(f'The mean of your data is {total/count:10.3f}')
```

```
Do you have more data (Y/N)yes
Enter a number: 6
Do you have more data (Y/N)yep
Enter a number: 7.5
Do you have more data (Y/N)y
Enter a number: 8.5
Do you have more data (Y/N)y
Enter a number: 9.3
Do you have more data (Y/N)n
The mean of your data is      7.160
>>>
```

This example adapted from John Zelle (2017) *Python Programming* (3rd Edition)

Using a sentinel loop for data entry

A better approach is to use a sentinel value - a data item value which is recognized as the end point of the data set being entered.
A good sentinel value is the empty string - detecting that just return was entered to signal the end of the data.

```
# program with sentinel loop
# to take data from the user and calculate mean

count = 0
total = 0

entryStr = input('Enter a number (press \'Enter\' to quit): ')
while entryStr !='':
    number = float(entryStr)
    total += number
    count += 1
    entryStr = input('Enter a number (press \'Enter\' to quit): ')

print(f'The mean of your data is {total/count:10.3f}')
```

```
Enter a number (press 'Enter' to quit): 4
Enter a number (press 'Enter' to quit): 5.6
Enter a number (press 'Enter' to quit): 6
Enter a number (press 'Enter' to quit): 7.2
Enter a number (press 'Enter' to quit):
The mean of your data is      5.700
```

This example adapted from John Zelle (2017) *Python Programming* (3rd Edition)

Using a while loop with break to check for valid data

A very common use of while loops is to check if data entry is valid before continuing.

break - is a Python reserved word to that breaks out of a loop before it has finished

```
while True:
    number = float(input('Enter a number between 1 and 10 : '))
    if number >= 1 and number <= 10:
        break
    else:
        print('Sorry that is out of range.')
```

```
Enter a number between 1 and 10 : 0
Sorry that is out of range.
Enter a number between 1 and 10 : 34
Sorry that is out of range.
Enter a number between 1 and 10 : 55
Sorry that is out of range.
Enter a number between 1 and 10 : 7
>>>
```

This example adapted from John Zelle (2017) *Python Programming* (3rd Edition)