University of
South Australia

Problem Solving and Programming

Week 3 – Control structures: while loops
– Sequences: Strings

Python Books

Course Textbook

Gaddis, Tony. 2012, *Starting Out with Python*, 2nd edition, Pearson Education, Inc.

Free Electronic Books

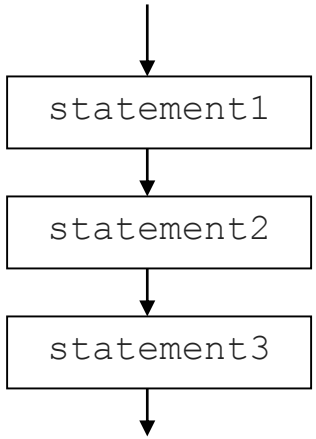
There are a number of good free on-line Python books. I recommend that you look at most and see if there is one that you enjoy reading. I find that some books just put me to sleep, while others I enjoy reading. You may enjoy quite a different style of book to me, so just because I say I like a book does not mean it is the one that is best for you to read.

- The following three books start from scratch - they don't assume you have done any prior programming:
  - The free on-line book "**How to think like a Computer Scientist: Learning with Python**", by Peter Wentworth, Jeffrey Elkner, Allen B. Downey and Chris Meyers, provides a good introduction to programming and the Python language. I recommend that you look at this book.
  - There is an on-line book "**A Byte of Python**" that is quite reasonable. See the home page for the book, or you can go directly to the on-line version for Python 3, or download a PDF copy of the book. This book is used in a number of Python courses at different universities and is another I recommend you look at.
  - Another good on-line book is "**Learning to Program**" by Alan Gauld. You can download the whole book in easy to print PDF format, and this is another book that would be good for you to look at.
- If you have done some programming before, you may like to look at the following:
  - **The Python Tutorial** - this is part of Python's documentation and is updated with each release of Python. This is not strictly an e-Book, but is book-sized.
  - **Dive into Python 3**, by Mark Pilgrim is a good book for those with some programming experience. I recommend you have a look at it. You can download a PDF copy.

UniSA

Control Structures

- Control Structures
  - Programs can be written using three control structures (Bohn and Jacopini):
    - **Sequence**
      Statements are executed one after the other in order.



    - **Selection** (making decisions)
      - `if`
      - `if-else`
      - `if-elif-else`
    - **Repetition** (doing the same thing over and over)
      - `while`
      - `for`

Repetition structures

- Loops are used when you need to repeat a set of instructions multiple times.

- `while` loop
  - Good choice when you need to keep repeating the instructions until a criteria is met.

Control Structures: `while` loop

- The `while` repetition structure.

- A `while` loop has the following form:

```
while expression:
    while_suite
else:
    else_suite
```

  - The `else` clause is optional.
  - As long as the `expression` is `True`, the while block's suite is executed.
  - If the `expression` is or becomes `False`, the loop terminates, and if the optional `else` clause is present, its suite is executed.

Control Structures: `while` loop

- An example:

```
a = 0
while a < 10:
    print(a)
    a = a + 1
else:
    print('done')
```

Important: must use consistent indentation – the start and end of blocks are indicated by indentation. Python uses indentation to signify its block structure.

Control Structures: `while` loop

- Same example without `else` clause – ***This style is preferred***:

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

print('done')
```

Important: must use consistent indentation – the start and end of blocks are indicated by indentation. Python uses indentation to signify its block structure.

Control Structures: `while` loop

- What is the output produced by the following code?

```
a = 0
while a < 3:
    print('going loopy', a)
    a = a + 1

print('yee ha!')
```

Control Structures: `while` loop

- What is the output produced by the following code?

```
a = 3
while a >= 0:
    print('In while loop', a)
    a -= 1

print('The end!')
```

An exercise…

- Let's revisit the following exercise…
  - Write a program to generate a random number between 1 – 10.
  - Display the random number to the screen as follows:

    Random number is: 7
  - Modify your program so that it asks (prompts) the user to guess the random number.
  - Display the user's guess to the screen.
  - Modify your program so that it displays 'Well done – you guessed it!' if the user guesses the number correctly.
  - Modify your program so that it displays 'Well done – you guessed it!' if the user guesses the number correctly, otherwise displays the message 'Too bad – better luck next time!' if the user guesses incorrectly.
  - Modify your program so that it displays 'Well done – you guessed it!' if the user guesses the number correctly, displays 'Too low' if the guess is lower than the random number, displays 'Too high' if the guess is higher than the random number.

UniSA

An exercise…

- Solution…

```
import random

number = random.randint(1, 10)
print('Random number is:', number)

guess = int(input('Please enter your guess: '))
print('You guessed:', guess)

if number == guess:
    print('Well done - you guessed it!')
elif guess < number:
    print('Too low')
elif guess > number:
    print('Too high')
```

An exercise…

Modify the program as follows:

- Add a while loop which allows the user to keep guessing until they guess the correct number.

An exercise…

Processing Input

- A `while` loop is commonly used to process input.
- Use a `while` loop to repeat a set of commands until a condition is met.
- When reading input, your loops should take the following form (algorithm).

```
prompt user for data                    # initialize loop control
get data

WHILE (value read in is OK)             # test loop control
    perform the processing

    prompt user for data                # update loop control
    get data
```

Processing Input

- Python example (read and sum numbers until the user enters a negative number):

```
total = 0

# initialise loop control
no = int(input('Please enter a number (-1 to quit): '))

# test loop control
while no >= 0:

    print('Number is: ', no)
    total = total + no

    # update loop control
    no = int(input('Please enter a number (-1 to quit): '))

print('Sum of all numbers entered is: ', total)
```

Processing Input

- Python example (read and display menu commands until the user enters 'q' to quit):

```
# initialise loop control
choice = input('Please enter [a, b, c or q to quit]: ')

# test loop control
while choice != 'q':

    print('Choice entered was:', choice)

    # update loop control
    choice = input('Please enter [a, b, c or q to quit]: ')


print('Thanks - we\'re done here!')
```

Processing Input

- Another common use for a `while` loop is error checking of user input.

Let's revisit an earlier example: Write a program that generates a random number between 1-10, asks the user to guess the number.

```
import random
number = random.randint(1,10)

# prompt for and read user's guess
guess = int(input('Please enter your guess: '))

# check to confirm that guess is in correct range – test loop control
while guess < 1 or guess > 10:
    print('Input must be between 1 – 10 inclusive.')

    # update loop control
    guess = int(input('Please enter your guess: '))


if number == guess:
    print('Well done- you guessed it!')
else:
    print('Too bad – better luck next time!')
```

Processing Input

- Another common use for a `while` loop is error checking of user input.

Let's revisit an earlier example: Write a program that reads and displays menu commands until the user enters 'q' to quit.

```
# prompt for and read user's choice
choice = input('Please enter [a, b, c or q to quit]: ')

# check to confirm that user enters either 'a', 'b', 'c' or 'q'
while choice != 'a' and choice != 'b' and choice != 'c' and choice != 'q':
    print('Input must be a, b, c or q to quit!  Please try again...')
    choice = input('Please enter [a, b, c or q to quit]: ')

# loop while user does not enter 'q'
while choice != 'q':

    # display user's choice to the screen
    print('Choice entered was:', choice)

    # prompt for and read user's choice
    choice = input('Please enter [a, b, c or q to quit]: ')

    # check to confirm that user enters either 'a', 'b', 'c' or 'q'
    while choice != 'a' and choice != 'b' and choice != 'c' and choice != 'q':
        print('Input must be a, b, c or q to quit!  Please try again...')
        choice = input('Please enter [a, b, c or q to quit]: ')

print('Thanks - we\'re done here!')
```

More on strings…

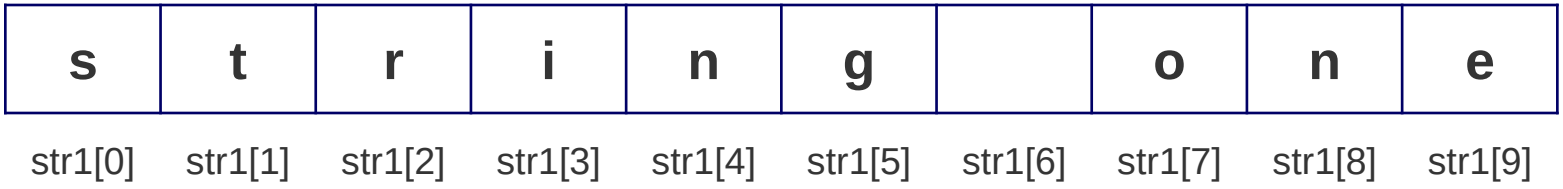
Sequences: more on strings…

- We can create a string by enclosing characters in quotes.
- Python treats single quotes the same as double quotes.
- Strings are immutable.
- Strings are made up of individual characters, which may be accessed via slicing.
- A string consists of a sequence of characters.
  - The first character of a string “hey” is ‘h’, the second character s ‘e’, etc.

Sequences: more on strings…

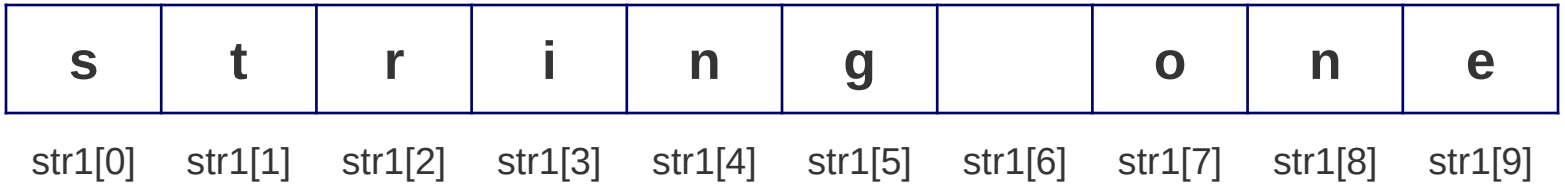
- Creating and assigning strings:

```
>>> str1 = 'string one'
>>> str2 = "this is a string"
>>> str3 = 'this is another string'
>>> print(str2)
this is a string
>>> str3
'this is another string'
```



UniSA

Sequences: more on strings…



- Accessing values (characters and substrings) in Strings:
  - Use square brackets for slicing along with the index or indices to obtain a substring.
  - We can access values using:

str1[start]	Value at index start. Eg: str1[2] has value 'r'
str1[start:end]	Slice from start to end-1. Eg: str1[2:6] is 'ring'
str1[start:end:step]	From start to end-1 with step size. Eg: str1[0:5:2] is 'srn'

UniSA

Sequences: more on strings…

- Accessing values (characters and substrings) in Strings:
  - Use square brackets for slicing along with the index or indices to obtain a substring.

```
>>> str1 = "this is a string"
>>> str1[0]
't'
>>> str1[5:9]
'is a'
>>> str1[5:]
'is a string'
>>> str1[:3]
'thi'
```

Sequences: more on strings…

- Comparison operators and strings:
  - Strings are compared lexicographically (ASCII value order).

```
>>> str1 = "abc"
>>> str2 = "abc"
>>> str3 = "xyz"
>>> str1 < str2
False
>>> str1 == str2
True
>>> str1 < str3
True
>>> str1 > str3
False
>>> str1 == str3
False
>>> str1 != str3
True
>>> str1 < str3 and str2 == "abc"
True
```

Sequences: more on strings…

- The `in` operator:
  - Useful for checking membership.
  - Returns a Boolean value – `True` or `False`.
  - For example:

```
>>> str1 = "aeiou"

>>> 'a' in str1
True

>>> 'z' in str1
False

>>> if 'u' in str1:
        print("It's a vowel!")

It's a vowel!
```

Sequences: more on strings…

- Sequence Operators:
  - Slices [] and [:]
  - The slice operator with a single argument will give us a single character.
  - The slice operator with a range (using a colon) will give us multiple consecutive characters.
  - Examples:

```
str1 = '123123456'
str1[0] = '1'
str1[1] = '2'
str1[:]     -> is the string str1 '123123456'
str1[:5]    -> is the first 5 characters of str1
str1[1:4]   -> is '231'
str1[5:]    -> is '3456'
str1[-1]    -> is '6'
str1[:3] + '-' + str1[6:]    -> is '123-456'
```

UniSA

Sequences: more on strings…

- Concatenation (+) operator
  - Create new strings from existing ones.

- Repetition (*) operator
  - Creates new strings, concatenating multiple copies of the same string.

```
>>> str1 = 'hi '
>>> str2 = 'there...'
>>> str1 + str2
'hi there...'
>>> str1 * 3
'hi hi hi '
```

UniSA

Sequences: more on strings…

- Updating strings:
  - Strings are immutable – changing an element of a string requires creating a new string.
  - Cannot change an existing string without creating a new one.
  - Cannot update individual characters or substrings in a string.
  - Update an existing string by (re)assigning a variable to another string.

```
>>> str1 = "this is"
>>> str2 = "a string"
>>> str1 = str1 + "something different"
>>> print(str1)
this issomething different
>>> str1 = "different altogether"
>>> print(str1)
different altogether
```

Sequences: more on strings…

- String Built-in functions:
  - len()
    - Returns the length of a string.
  - max()
    - Returns the greatest character.
  - min()
    - Returns the least character.

```
>>> str1 = "roger"
>>> len(str1)
5
>>> max(str1)
'r'
>>> min(str1)
'e'
```

Sequences: more on strings…

- String methods:
  - Strings are objects – have data and processing for data. Processing via methods that are called using 'dot notation'.

  - For example:

    if `str1` is a string, then

    ```
    str1.upper()
    ```

    is a method that returns a string that has characters of `str1` converted to upper case.

    ```
    >>> str1 = 'this is fun'
    >>> str1.upper()
    'THIS IS FUN'
    ```

Sequences: more on strings…

- Some useful string methods:
  - string.lower()

    converts all uppercase letters in string to lowercase.
  - string.upper()

    converts all lowercase letters in string to uppercase.
  - string.find(t, start, end)

    find string t in string or in substring if given [start:end].
  - string.split(sep, maxsplit)

    returns a list of the words in the string, using sep as the delimiter string (default is space). If maxsplit is given, at most maxsplit splits are done.
  - See Python docs for all string methods.

Sequences: more on strings…

- Some programming tasks require you to access the individual characters in a string.
- Iterating over a String with the while loop:
  - Access the individual characters in a string with an index. Each character in a string has an index that specifies its position in the string.
  - To do so, we use the `len()` function as follows:

```
str1 = 'kramer'
index = 0
while index < len(str1):
    print(str1[index], end=' ')
    index = index + 1
```

The `len()` function prevents the loop from iterating beyond the end of the string. The loop iterates as long as index is less than the length of the string.

An exercise…

- What is the output produced by the following code?

```
str1 = 'kramer'

index = 0
while index < len(str1):
    print(str1[index], end=' ')
    index = index + 1

print('\n\nThe End!')
```

Output: ??

An exercise…

- What is the output produced by the following code?

```
str1 = 'kramer'

index = 0
while index < len(str1):
    print(str1[index], end=' ')
    index = index + 1

print('\n\nThe End!')
```

Output:

```
k r a m e r


The End!
```

UniSA

An exercise…

- What is the output produced by the following code?

```
str1 = 'Over The Top'
new_string = ''

index = 0
while index < len(str1):
    if str1[index].isupper():
        new_string += str1[index]

    index = index + 1

print(new_string)
```

Output: ??

An exercise…

- What is the output produced by the following code?

```
str1 = 'Over The Top'
new_string = ''

index = 0
while index < len(str1):
    if str1[index].isupper():
        new_string += str1[index]

    index = index + 1

print(new_string)
```

Output:

```
OTT
```

UniSA

End of Week 3