COMPSCI 101
Principles of Programming

Lecture 24 - Using the Python interpreter, Python sequences

Learning outcomes

- At the end of this lecture, students should be able to:
  - recognise sequences and the common features of sequences
  - use the interactive Python interpreter to check python statements and functions
  - use the interactive Python interpreter to look up Python help

Sequences: strings, lists and tuples

- Sequence types
  - There are five types of sequences in Python. In CompSci 101 we use three of these: strings, lists and tuples.

```
a_tuple = (3, 4, 8)
a_list = [3, 4, 8]
a_string = "348"
```

  - Sequences allow you to store multiple values in an organized and efficient fashion.
  - The indices of the elements of a sequence start at 0. The indices can be negative (to access elements from the end of the sequence).
  - The order of the elements in a sequence is important.
  - Each element of a sequence can be accessed using square brackets and the index number, e.g.,

```
a_tuple = (3, 4, 8)
print(a_tuple[2])
middle = a_list[1]
last = a_string[-1]
```

Sequences continued

- Sequences can be sliced:

```
a_tuple = (3, 4, 8, 7, 2)
a_list = [3, 4, 8, 0, 1]
a_string = "3and 4"
a_tuple2 = a_tuple[0:3:2]
a_list2 = a_list[1:3]
print(a_tuple2, a_list2, a_string[5:1:-2])
```

```
(3, 8) [4, 8] 4d
```

- The len(), min(), max() functions can be applied to sequences (sum() can be used with tuples and lists).

```
a_tuple_list = [(3, 'c'), (9,'a'), (1, 'z')]
print(len(a_tuple))
print(len(a_string))
print(max(a_tuple))
print(max(a_string))
print(len(a_tuple_list))
print(max(a_tuple_list))
```

```
5
6
8
n
3
(9, 'a')
```

Sequences continued

- The +, *, and 'in' operators can all be used with sequences

```
a_tuple = (3, 4) * 3 + (2, 1)
a_list = [3, 0, 1] + [6, 2] * 2
a_string = "3 & 4" * 2 + "end"
print(a_tuple)
print(a_list)
print(a_string)

print(4 not in a_tuple, 24 in a_list, "23" in a_string)
```

```
(3, 4, 3, 4, 3, 4, 2, 1)
[3, 0, 1, 6, 2, 6, 2]]
3 & 43 & 4end
False False False
```

Iterating through the elements of sequences

- A for … in … loop can be used to visit each element of a sequence, e.g.,

```
a_tuple = (3, 4, 8, 7, 2)
a_list = [3, 4, 8, 24, 1]

total = 0
for number in a_tuple:
    total += number
print("1.", total)

total = 0
for number in a_list:
    total += number
print("2.", total )
```

```
1. 24
2. 40
```

Iterating through the elements of strings

- A for … in … loop is used to visit each character in a string sequence. The elements of a string sequence are the characters making up the string.

```
word = "wonderful"
number = 0

for letter in word:
   if letter in "aeiou":
       number += 1

print(number)
```

```
3
```

Iterating through the characters of a sequence – Exercise 1

- Complete the **`get_num_uniques()`** function which returns the number of unique elements in the sequence (including non alphabetic characters).

```
def get_num_uniques(a_sequence):
    uniques = []
    for
        if
          uniques.
    return len(uniques)

def use_get_num_uniques():
    words = "Number of unique elements:"
    print(words, get_num_uniques("green apple"))
    print(words, get_num_uniques("abcdefg") )
    print(words, get_num_uniques("abbbbbb") )
    print(words, get_num_uniques((3, 4, 3, 3, 4, 6, 3, 7, 8, 4)) )
    print(words, get_num_uniques([3, 4, 3, 3, 4, 6, 3, 7, 8, 4]) )
main()
```

```
Number of unique elements: 8
Number of unique elements: 7
Number of unique elements: 2
Number of unique elements: 5
Number of unique elements: 5
```

Iterating through the characters of a string – Exercise 2

- Complete the count_longer_words() function to find the count of words that are longer than the parameter word from a given list of words.

```
def count_longer_words(a_list, word):
   count = 0
   for
      if

   return count

def main():
   print(count_longer_words(['Double', 'letters', 'in', 'green', 'apple'], 'go'))
    print(count_longer_words(['Number', 'of', 'unique', 'elements'], 'go'))
main()
```

```
4
3
```

Iterating through the characters of a string – Exercise 3

- Complete the **count_doubles()** function which returns the number of double letters (a letter followed by the same letter) excluding double spaces, in the string passed as a parameter.

```
def count_doubles(text):
   count = 0
   ...


def main():
   print("Double letters in green apple", count_doubles("green apple"))
   print("Double letters in abcdefg", count_doubles("abcdefg"))
   print("Double letters in abbbbbb", count_doubles("abbbbbb"))"

main()
```

```
Double letters in green apple 2
Double letters in abcdefg 0
Double letters in abbbbbb 3
```

Compilers and interpreters

- Compilers
  - Compilers convert source code into machine code and store the machine code in a file. The machine code can then be run directly by the operating system as an executable program (… .exe file).

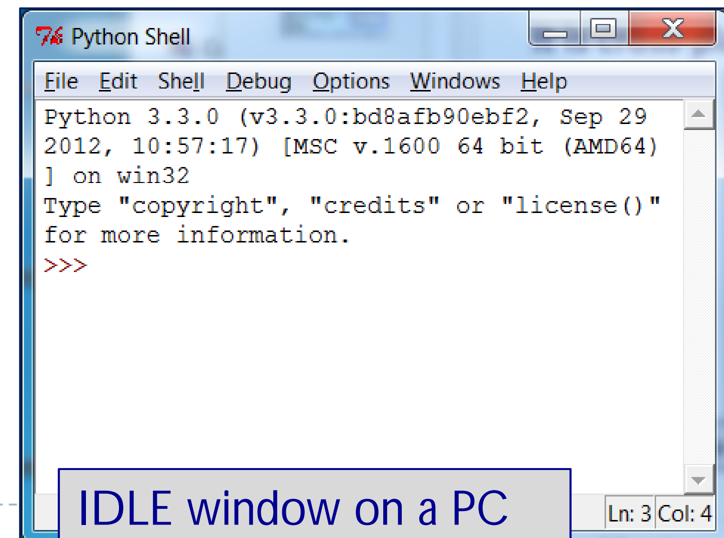
- Interpreters
  - Interpreters bypass the compilation process and convert and execute the code directly statement by statement.
  - Python is an interpreted language, i.e., the Python interpreter reads and executes each statement of the Python source program statement by statement:
    - this is why even if you can have an error in the program further down, the program executes until it hits that error.

Python IDLE

- IDLE (Integrated DeveLopment Environment) is an integrated development environment for Python. This is the development environment provided when you download Python.
  - WIKIPEDIA states "IDLE is intended to be a simple IDE and suitable for beginners, especially in an educational environment. To that end, it is cross-platform, and avoids feature clutter."

IDLE provides an interactive environment for checking Python code and for running Python programs.

IDLE window on a MAC



IDLE window on a PC

The Python interactive interpreter (Python shell)

- The interactive Python interpreter
  - The Python interactive interpreter makes it easy to check Python commands.
  - Open the interactive interpreter
    - We will use IDLE which opens a window with the interpreter prompt: >>>
    - Once the Python interpreter has started any Python command can be executed (at the prompt >>>)

```
>>> word = "amazing"
>>> len(word)
7
>>> word = word * 3
>>> word
'amazingamazingamazing'
>>> another_word = word[2::3]
>>> another_word
'anmiazg'
>>> word[:0:-4]
'ganmi'
```

Notice that the interpreter displays the result of each statement even though there is no print() in the statement.

The Python interactive interpreter cont.

- The interactive Python interpreter can also be used to test functions
  - The Python interactive interpreter makes it easy to check Python code.

```
>>> def get_result(command, what_to_do, where):
        return command + " " + what_to_do + " in the " + where

>>> get_result("a", "b", "c")
'a b in the c'
>>> get_result("come", "sing", "hall")
'come sing in the hall'
>>> get_result("go", "jump", "pond")
'go jump in the pond
```

Notice that it is necessary to insert a blank line to end the function definition.

See the results of calling the function three times with different arguments.

The Python interactive interpreter help

- The interactive Python interpreter can also be used to get help:

```
>>> help(str.rfind)
rfind(...)
    S.rfind(sub[, start[, end]]) -> int
    Return the highest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice
    notation.
    Return -1 on failure.

>>> help(sum)
sum(...)
    sum(iterable[, start]) -> value

    Return the sum of an iterable of numbers (NOT strings) plus
    the value of parameter 'start' (which defaults to 0).  When
    the iterable is empty, return start.
```

None

- print statements (in the interpreter window) just print to the interpreter window.
- A function which does not explicitly return a value, always returns None.

```
>>> def do_little(n1, n2):
        print("Sum:", n1 + n2)

>>> do_little(3, 5)
Sum: 8
>>> print(do_little(3, 5))
Sum: 8
None
```

Notice that it is necessary to insert a blank line to end the function definition.

The code in the function executes.

The result of calling the function is printed.

Summary

- strings, lists and tuples are sequences
  - The operators: +, * and in can be used with sequences
  - We use a for … in … to iterate through each element of a sequence
  - len(), min(), max() can be used with sequences
  - sum() can be used with tuples and lists
  - Each element of a sequence can be accessed using the index operator. The index can be negative (starting from the end of the sequence)
  - Sequences can be sliced using [slice_start: slice_end: step]
- The Python interactive interpreter (IDLE)
  - use the interactive Python interpreter to check python statements and functions
  - use the interactive Python interpreter to look up Python help