Fundamentals of Programming 1

Lecture 10

Aurelia Power, ITB, 2018

More on Strings

- As we have already learnt, strings are sequences of characters: for instance, the string 'Aurelia' can be viewed as an ordered sequence of characters:

A	u	r	e	l	i	a

- Sequences typically are structures that contain items in a certain order.
- For instance, a string can be viewed as a sequence of characters where each character is an item
- Each item (character) has a certain position/index in the string and it can then be accessed by its place in the sequence
- **For example**, 'A' is the first item/character in the string, 'u' is the second item/character in the string and so on…

More on Strings

- Because strings are ordered sequences, each item/character can be accessed by its position/index in the string
- However, indexing of strings and other sequences in Python starts at 0 = aka **zero-based indexing**. (Note: python also uses negative indices, not covered here).
- EXAMPLE:

A	u	r	e	l	i	a
0	1	2	3	4	5	6

characters

indices

- So, the first character in any string is at index **0**: 'A' is at index 0
- The second character is at index **1**: 'u' is at index 1
- And so on…
- So, the last character is at index **len(string) -1**: last 'a' is at index 6 = len('Aurelia') – 1= 7 -1

More on Strings

- We can use the indices/positions of the characters to access individual characters
- To do so we need to use a pair of square brackets [] as follows:

string [index]

- EXAMPLE: my_string = 'Aurelia'

A	u	r	e	l	i	a
0	1	2	3	4	5	6

```
print ( my_string [ 0 ] ) ##  will output 'A'
print ( my_string [ 1 ] ) ##  will output 'u'
print ( my_string [ 4 ] ) ##  will output 'l'
print ( my_string [ len(my_string) -2] )
##  will output 'i'
```

More on Strings

- We can iterate over a string's elements/characters using a for loop :

```
str1 = 'Python'
for character in str1:
    print(character, end=' ')
## outputs each character separated by a space: P y t h o n

str2 = 'Anaconda'
count = 0
for c in str2:
    if c == 'a':
        count += 1
print ( count ) ## what does it output??
## 2 because there are two a's in Anaconda
```

More on Strings – your turn…

- Given the string 'I love FOP1', answer the following questions:

What is the character at index 1?

What is the character at index 7?

- What is the purpose of the following function (assume s is a string)?

```
def mystery_function(s):
    count = 0
    for x in range(len(s)):
        if s[x] == 'a' or s[x] == 'e' or s[x] == 'i' or s[x] == 'o' or s[x] == 'u':
            count += 1
    return count
```

- What kind of function is it? Call the above function twice.

More on Strings

- We can also access consecutive characters/items (aka **substring**) using slicing.
- To do so we need to use a pair of square brackets [] and the colon : as follows

string [some_index : another_index]

- this will provide a string formed by all characters starting from some_index (included) to another_index (not included)
- EXAMPLE: my_string = 'Aurelia'

A	u	r	e	l	i	a
0	1	2	3	4	5	6

print (my_string [0 : 3]) ## will output 'Aur' – all characters from 0 (included) to 3 (not included) because at index 3 is 'e' but that's not included

print (my_string [2 : 4]) ## will output 're'

print (my_string [1 : len(my_string) -1])

will output 'ureli'

More on Strings

- So when using slicing, the start index is always included, while the end is always excluded
- We can omit the start index: **string [: another_index]** – in which case the start index will default to **0**
- We can also omit the end index: **string [some_index :]** – in which case the end index will default to **len(string)**
- EXAM

A	u	r	e	l	i	a
0	1	2	3	4	5	6

print (my_string [: 3]) ## will output 'Aur' – all characters from 0 (included) to 3 (not included) because at index 3 is 'e' but that's not included

print (my_string [2 :]) ## will output 'relia'

print (my_string [:])

will output 'Aurelia' since both the start and the end indices are omitted, so they will both default

More on Strings – your turn…

- What is the output of the following code?

```
my_var = 'John' + ' and ' + 'Mary'
print(my_var[0:5])
print(my_var[len(my_var - 5) : ])
print(my_var[: 3] + my_var[3 : ])
```

- What is the purpose of the following function (assume s1 is a string)?

```
def  some_function(s1):
    x = 0
    y = len(s1) - 1
    while x <= y:
        if s1[x] != s1[y]:
            return False
        x += 1; y -= 1
    return True
```

- Call this function 3 times.

An Introduction to Python Lists

- So far, we only worked with individual numerical and string values.
- But many times we need access to multiple values stored as a collection of items, for instance a collection of grades or a collection of student names

- One obvious way to organise and store multiple values is a list
- For instance a shopping list, or human DNA which is a list of molecules.

- **A list** = a linear data structure that is ordered according to its items location.

- The location/position of a list item is called an **index** (same as in a string)
- For example, let's assume that there are 7 students in group 7 and they got the following grades in MCQ2: 10, 15, 12, 10, 10, 7, 3 – these grades can be stored in a list that can be visualised as a sequence:

10	15	12	10	10	7	3

An Introduction to Python Lists

NOTE: Like with strings, list indexing starts at 0 → so the last index is the number of elements/items -1

10	15	12	10	10	7	3
0	1	2	3	4	5	6

- So at index 0 we have 10, at index 5 we have 7, etc.

- Lists can have duplicate items: there are three 10 values in the above list

- There are several ways to create lists in Python:

```
my_list = [ ] ## an empty list
grades = [10, 15, 12, 10, 10, 7, 3] ## a list with 7 integers
```

- Lists can contain same type items as in *grades*
- But they can contain items of different data type:

```
mixed_list = ['a', 3, 7.1, False, 'True']
```

Common List Operations

- Unlike strings, lists are sequences that can be modified, that is, they are **mutable**
- For example, to the list below we can add other values, or remove values…

10	15	12	10	10	7	3
0	1	2	3	4	5	6

- As a result, we can perform various operations on a list.

➢ **Retrieving elements** of a list is similar to the way we access characters in a string:
We can use the indices to access its elements:

```
print(grades[2]) ##12
print(grades[5]) ## 7
```

➢ **Updating the values/elements** of a list: for instance, I realised that last grade entered is the wrong one, it should be 13 → to modify it we can use assignment:

```
print(grades) ## [10, 15, 12, 10, 10, 7, 3]
grades[6] = 13
print(grades)
## [10, 15, 12, 10, 10, 7, 13]
```

Common List Operations

- Using the same list as on the previous slide: [10, 15, 12, 10, 10, 7, 13]

Inserting elements/items into a list is achieved using the method *insert;* for instance, I forgot to enter a grade after the first 10:

```
grades.insert(1, 4) ## places at index 1 the grade 4 and then shifts the other
                    ## grades one place to the right
print ( grades ) ## [10, 4, 15, 12, 10, 10, 7, 13]
print( len(grades) )  ## 8, because we have 8  grades/items now
```

Removing elements/items of a list can be achieved using thde del operator

```
del grades[2] ## removes the third item which is 15
print ( grades ) ## [10, 4, 12, 10, 10, 7, 13]
```

Appending elements/items to a list means adding them at the end of the list

For instance, using the grades list, we can add few more grades at the end:

```
grades.append ( 7 )
grades.append ( 5 )
grades.append ( 7 )
print ( grades ) ## [10, 4, 12, 10, 10, 7, 13, 7, 5, 7]
```

Iterating over List Elements/Items

- Using the same list as the previous slide: grades = [10, 4, 12, 10, 10, 7, 13, 7, 5, 7]
- We can use the for loop and its loop variable to access (but not modify/alter) the elements of the list, for instance, when we need to calculate average:

```
total = 0
for k in grades: ## in each iteration the variable loop k will take a different value;
                 ## for instance, in first iteration k is assigned the value of the first
                 ## item in the list grades which is 10; in the second iteration k is
                 ## assigned 4 and so on… until the entire list is exhausted
        total += k
print( total/len(grades) ) ## 8.5
```

- We can use the index variable also defined as a for loop variable:

```
total = 0
for k in range(len(grades)):
        total += grades[k] ## retrieve the item from the list using the index k and add
                           ## it into the total
print( total/len(grades) )  ## print the average value
```

Iterating over List Elements/Items

- We can use the index variable and the while loop:

```
total = 0 ;   k = 0
while k < len(grades): ## in each iteration the variable loop k will take a
                       ## different value; for instance, in first iteration k is
                       ## assigned the value of the first item in the sequence
                          ## generated by the range function: 0, and so on…
        total += grades[k] ## retrieve the item from the list using the index k and add
                           ## it into the total
        k += 1 ## increase k by 1 in each iteration
 print( total/len(grades) ) ## print the value of average
```

- Another example:

```
names = ['anna', 'john', 'helen', 'brendan', 'jack']
for name in names:
     if 'j' in name:
          print(name)
## it outputs john and jack on different lines, because they are the only names that
## contain 'j'
```

Creating custom lists

- We can use fill a list with selected values
- For instance, assume that from a list of grades I only want to get only those grades that are 10 or over:

```
grades = [10, 4, 15, 12, 10, 15, 10, 7, 13, 7, 5, 7, 13, 2, 7, 6, 8, 12, 15, 7, 6, 6, 8, 9, 10, 2]
grades_over_9 = [] ## initially the list is empty
for grade in grades: ## going through each grade in the list grades
    if grade > 9: ## if the selected grade is over 9
        grades_over_9.append(grade) ## we add that grade to the grades_over_9
print(grades_over_9) ## we now output the new list
```

- Another example: we want to buy only the healthy food items

```
foods_in_shop = ['pizza', 'oranges', 'lettuce', 'milk', 'cheese', 'fish', 'icecream', 'cocacola',
'chicken', 'bananas', 'broccoli', 'carrots', 'onions', 'cake', 'strawberries', 'yoghurt', 'kiwi',
'peppers', 'ham', 'doughnuts']
junk_foods = ['pizza', 'icecream', 'cocacola','cake','doughnuts']
my_basket = []
for food in foods_in_shop:
    if food not in junk_foods:
        my_basket.append(food)
print(my_basket)
```

Python Lists - your turn …

- What does the following code fragment output?

```
chars = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
for char in chars:
    if char in ‘anaconda’:
        print(char, end=‘ ‘)
```

- What does the following code fragment output?

```
nums = [34, 12, 55, 21, 77, -11, -10, 16, 23, 25]; x = 2
for num in nums:
    if num % x == 1:
        print(num * ‘%’)
```

- What does the following code fragment output?

```
nums = [34, 12, 55, 21, 77, -11, -10, 16, 23, 25]; x = 0; items = []
for num in nums:
    if num >= 12 and num <= 61:
        items.append(num)
print(items)
```

Python Lists - your turn …

- What is the purpose of the following function?

```
def f1 (x):
    y =[]
    for k in range(len(x)):
        if x[k] not in y:
            y.append(x[k])
    return y
```

Call this function several times.

- What is the purpose of the following function?

```
def f2 (x):
    y =['a', 'e', 'i', 'o', 'u']
    for k in x:
        if k not in y:
            print('…')
```

Call this function several times.