Unit 10. Moving Forward

Héctor D. Menéndez

Endless Science

endsci.net

Index

1 Moving Forward Concepts

2 Breaking The Control Flow

3 Exceptions

4 Structured Files

5 Exercises

Index

1 Moving Forward Concepts

2 Breaking The Control Flow

3 Exceptions

4 Structured Files

5 Exercises

New Concepts

There are a few concepts we will work on before we end this block:

- The control flow can be broken.
- We can make decisions based on errors.
- We can manipulate files that follow a specific structure.

Index

1 Moving Forward Concepts

2 Breaking The Control Flow

3 Exceptions

4 Structured Files

5 Exercises

Manipulating The Control Flow

A loop normally runs while its condition is satisfied.

In structural programming this is the only "legal" way to end a loop.

However, there are two instructions that can manipulate the way loops run without affecting the condition: `break` and `continue`.

Breaking The Control Flow

If we want to stop a loop before the condition is unsatisfied we can use the instruction `break`.

The instruction `break` will stop the loop.

```
1 for i in range(0,10):
2   if i == 5:
3     break
4 print(i)
```

Listing 1: break

Continuing The Control Flow

If we want to skip a step in a loop we can use the instruction `continue`.

The instruction `continue` will end the current step and jump to the next one.

```
1 for i in range(0,10):
2   if i == 5:
3     continue
4   print(i)
```

Listing 2: continue

Do Nothing

If we want to indicate that a block is pending within a code, we can use the statement `pass`.

The instruction `pass` will do nothing but will allow a block to be created.

```
1 for i in range(0,10):
2   if i == 5:
3     pass
4   print(i)
```

Listing 3: pass

Index

1 Moving Forward Concepts

2 Breaking The Control Flow

3 Exceptions

4 Structured Files

5 Exercises

Exceptions

Traditionally, when a function failed, it used to return an error code, for example, imagine the following division of positive numbers:

```
1 def positiveDiv(num1,num2):
2   if(num1 < 0 ):
3     return -1
4   if(num2 < 0):
5     return -2
6   if(num2 == 0):
7     return -3
8   return num1/num2
```

Listing 4: Positive Division

Checking whether the result is negative or not will tell whether the function worked or not. However, if the function is a division for positives and negatives, this becomes tricky.

Exceptions

To solve this problem we can use the concept of `exception`.

When an error is reached, the program can throw an exception indicating the type of error.

Some common situations that lead into an exception are: division by zero, getting an element of a list that does not exist, transforming a string that is not an integer into an integer, etc.

Catching Exceptions

An `exception` will end a program but this mechanism also gives an opportunity to recover from the error. This mechanism is called catching an exception.

Try the following program. It will stop, throwing an exception or error, once it reaches the division by zero:

```
num1=2
num2=0

num1/num2
print(num1)
```

Listing 5: Division by Zero

Catching Exceptions

Now we are going to recover from that error in two steps. The first step will `try` to run the potentially erroneous line.

If everything goes well, nothing will happen and the program will go on. However, if something goes wrong we will catch the exception and print an error message, and the program will continue.

```
1 num1=2
2 num2=0
3 try:
4   num1/num2
5 except:
6   print("You tried a division by zero")
7 print(num1)
```

Listing 6: Division by Zero

Exception Types

More than one exception can be associated with an instruction. We can specify the type of exception that we want to catch and specify different actions depending on their nature.

Exception	**Description**
KeyboardInterrupt	The user send an interruption signal (Ctrl+C).
TypeError	Incorrect variable type for a specific operation.
NameError	The variable name is not in the local scope.
IOError	An input/output device raised an error.
ZeroDivisionError	Division by zero.
IndexError	The index is out of the scope of the list.
KeyError	The key is not part of the dictionary.
MemoryError	Out of memory.
RuntimeError	No idea, but something happened.

Exception Types

We can specify the type of exception we are handling after `except`.

```
1 list=[1,2,0,4]
2 try:
3   list[5]/list[2]
4 except IndexError:
5   print("You tried to read an index outside of the list
      ")
6 except ZeroDivisionError:
7   print("You tried to divide by zero")
8 print(list)
```

Listing 7: Double Exception

We will see more about exception and how to define our own during objects.

Index

1 Moving Forward Concepts

2 Breaking The Control Flow

3 Exceptions

4 Structured Files

5 Exercises

Structured Files

Some files are neither binary or text files but special files that follow a specific structured.

Some good examples are XML files (for example .docx or .xlsx), JSON files (normally used in network communications) and CSV or ARFF files (normally used in data analysis).

Structured files normally follow a standard and they are parsed before they are read.

Structured Files: Example of CSV Files

CSV files are frequently used for tables. They represent rows and columns.

In a CSV file every line is a row and the columns are separated with a character, normally a ',', ';' or TAB.

Imagine the file marks.csv:

```
1 Student;Mark CW 1;Mark CW 2;Total
2 John;10;10;20
3 Anna;10;10;20
4 Fred;5;10;15
```

Listing 8: marks.csv

Structured Files: Example of CSV Files

We can use the library pandas to manage CSVs.

```
import pandas as pd
csvFile = pd.read_csv("marks.csv",sep=';')#Needs the
    separator
print(csvFile)
print(csvFile[["Student","Total"]])#Shows only 2
    columns
print(csvFile[csvFile["Total"]>17])#Shows values under
    a condition
print(csvFile.iloc[1:3,0:3])#Shows specific ranges of
    rows and columns
simplerCSV=csvFile[["Student","Total"]] #Creates a
    reduces version
print(simplerCSV)
simplerCSV.to_csv("simplerMarks.csv",sep=';') #Saves
    into a file
```

Listing 9: Managing CSVs

Structured Files: Open XML

Open XML is a standard format for documents normally under the extension .docx, .xlsx and .pptx.

Python has libraries to read and write Open XML documents. We will see two examples: docx and xlsx.

For these we need to install the libraries in conda. In your terminal type:

```
conda install -c conda-forge python-docx
conda install -c anaconda xlrd
```

Listing 10: OpenXML libraries

Structured Files: Creating a DOCX File

Let's write a docx file with Python:

```
1  from docx import Document
2
3  doc = Document() #creates the document
4  doc.add_heading('This is the Title')
5  doc.add_heading('The Subtitle', level=2)
6  pgraph = doc.add_paragraph('A simple paragraph.')
7  pgraph.insert_paragraph_before('The first paragraph')
8  doc.add_page_break()
9  tab = doc.add_table(rows=2, cols=2)
10 for row in tab.rows:
11     for cell in row.cells:
12         cell.text="Cell"
13 doc.save("mypython.docx")
```

Listing 11: Writing docx

Structured Files: Reading a DOCX File

Let's read a docx file with Python:

```
1  from docx import Document
2
3  doc = Document("mypython.docx") #opens the document
4
5  for paragraph in doc.paragraphs:
6      print(paragraph.text)
7
8  table = doc.tables[0]
9  for row in table.rows:
10     for cell in row.cells:
11         print(cell.text)
12         cell.text = "Other cell" #Changes the cell's
       text
13
14 doc.save("modified.docx")
```

Listing 12: Reading docx

Structured Files: Creating a XLSX File

Let's write a xlsx file with Python:

```
1  from openpyxl import Workbook
2  from openpyxl.comments import Comment
3
4  workbook = Workbook()
5  currentSheet = workbook.active
6  currentSheet.append(["Student","CW1 Mark","CW2 Mark","
       Total"]) #Appends rows
7  currentSheet.append(["John",10,10])
8  currentSheet["A3"]="Anna" #We modify a cell
9  comment = Comment("Our best Student","") #We add a
       comment to Anna
10 currentSheet["A3"].comment = comment
11 currentSheet["D2"]="=SUM(B2:C2)" #We put a formula for
       the total
12 workbook.save("sample.xlsx")
```

Listing 13: Writing xlsx

Structured Files: Reading a XLSX File

Let's read a xlsx file with Python:

```
1  from openpyxl import load_workbook
2
3  workbook = load_workbook(filename = 'sample.xlsx')
4  currentSheet = workbook[workbook.sheetnames[0]]#We get
       the 1st sheet
5  for row in currentSheet.rows:
6      for cell in row:
7          if(cell.value):
8              print(cell.value)
9          if(cell.comment):
10             print(cell.comment.text)
11             cell.comment.text="Even better than before"
       # We only modify the comment
12 workbook.save("modified.xlsx")
```

Listing 14: Reading xlsx

Index

1 Moving Forward Concepts

2 Breaking The Control Flow

3 Exceptions

4 Structured Files

5 Exercises

Exercise 1: Preparing for the Use Case

Imagine that you use a xlsx file to put your student's mark, but you are required to pass those marks to reports in docx. The marks are divided in different parts (for example, implementation, quality of code, design, etc). You need to provide no only the mark but also a comment for every mark within the report.

Think about how you would organise your xlsx file and which modifications will be require in the docx file to be able to create a program that passes the marks from the xlsx file to the docx file.