Flow Control: Branches and loops

In this context flow control refers to controlling the flow of the execution of your program – that is, which instructions will get carried out and in what order. In the simplest of scripts, the instructions will get carried out in sequence starting from the top and running one by one until all the instructions have been executed.

In more sophisticated scripts and programs you might need to have more complex structures in your program. There are three key types of flow control structure in Python: loops, branches, and functions.

This document concerns itself with the first two of these – branches and loops.

Branches
The execution of your program branches when it reaches a decision point where it conditionally executes one bit of code or another. This is implemented in Python using the if-statement:

```python
a = 10
if a < 10:
    print('a is less than 10')
```

The if-statement consists of the word if followed by a condition terminated with a colon. The next line must be indented, in PyCharm the editor will do this for you. The number of spaces doesn’t matter but 4-spaces is a common choice. All the following lines with the same level of indentation are considered part of the if-block. The block continues until it reaches a line with a different level of indentation. The statements in the block will only be executed if the expression after the word if evaluates to true.

```python
# get input from the console and convert it to an integer
a = int(input())
if a == 10:
    print('a is equal to ten')

print('this is not in the if blocks')
```

The lines following the if statement with the same level of indentation are part of the if block – and will only execute if the expression in the if-statement is true.
You can also supply alternative code to execute if the expression is false using the “else” keyword:

```python
# get input from the console and convert it to an integer
a = int(input())
if a == 10:
    print('a is equal to ten')
else:
    print('a is not equal to ten')
print('this is not in the if or else blocks')
```

In this example, it will print the first message only if the expression is true, otherwise it will print the second message. The third message is not in the if or else blocks and so will always be printed.

You can even chain together alternative tests in the following way using the “elif” keyword (this is shorthand for “else if”):

```python
# get input from the console and convert it to an integer
a = int(input())
if a == 10:
    print('a is equal to ten')
elif a < 10:
    print('a is less than ten')
elif a > 10:
    print('a is greater than ten')
else:
    print('this should never print!')
print('this is not in the if, elif or else blocks')
```

In this case each test is executed in order until it reaches the first one to match – if none match then it will execute the else-block.

**Loops**

Loops are another type of flow control that involve repetition of instructions. There are two kinds of loop in Python – the for-loop and the while-loop.

**For-loops**

The for-loop is used to operate on sequences with a set of instructions for each element of the sequence.

Given a list of numbers the following code would loop over all numbers in the list in order to add them up:
In this code, first a list is created and stored in a variable called numbers. Then a variable called total is initialized to 0 – this is where the sum of all the numbers will be stored. Then comes the for-loop.

The for-loop repeats everything in the indented block for each element of the sequence. It starts with the first element and assigns its value to the variable ‘i’. It then carries out the instructions in the block. When those instructions are finished it moves on to the next element of the sequence, assigning the value to the variable ‘i’ and executes the instructions again. The loop keeps going until it have processed every element of the sequence.

A special case of a for-loop is when you just want to repeat something a set number of times:

```python
for i in range(10):
    print(i, 'Hello')
```

In this case the code makes use of the range() function. The function range(a,b) creates an arithmetic progression from a to b – including a but excluding b. So the above code will print the following to the output:

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

If you want to loop over a subset of the element of a sequence you can either use the range() function with the appropriate values or use a slice (remember from the document N_3_BasicDataStructures.pdf).

An example of this might be the following which prints out the last three elements of the list:

```python
li = ['a', 'b', 'c', 'd', 'e', 'f']
for i in li[-3:]:
    print(i)
```
While loops
The second kind of loop is the while loop. The while loop is used to continue executing the same instructions over and over again until certain specific conditions are met. A while loop looks like this:

```python
a = 0
while a == 0:
    a = int(input('Enter a non-zero value for a to exit the loop:'))
```

In this example, a variable is created called `a` and set to the value 0. The loop then begins with the keyword ‘while’ followed by a condition and a colon. The indented code on the following lines is repeatedly executed until the condition in the while loop is false. The code in the loop here reads a value from the console, converts it to an integer and stores the result in the variable `a`. The loop will keep asking you to enter a value until you enter something other than 0. When that occurs the test in the while-statement will become false and the loop will exit.

Breaking the loop
If for some reason you need to break out of a loop there are two important keywords: continue and break. If you put the word continue inside a loop then when it gets executed the loop will stop the current iteration where it is without executing anything after the continue statement and go on to the next iteration of the loop. If you use the break statement inside a loop – then when it gets executed it will break out of the loop completely and continue executing from the first instruction after the loop.

```python
a = 10
while True:
    a = int(input('Enter a value for a'))
    if a == 0:
        break
```

This code demonstrates the use of break – it also demonstrates the use of True – which is a value that always gives a true result for a test. So this loop is endless. The only way out of the loop is via the break statement. This will only get executed when `a == 0`. 