1 Flow Control, Loops and Exception Handling

1.1 13.2 if . . . elif . . . else

if . . . elif . . . else blocks always begin with an if statement immediately followed by a scalar logical expression. elif and else are optional and can always be replicated using nested if statements at the expense of more complex logic and deeper nesting. The generic form of an if . . . elif . . . else block is

```
if logical_1:
Code to run if logical_1
elif logical_2:
Code to run if logical_2 and not logical_1
elif logical_3:
Code to run if logical_3 and not logical_1 or logical_2
...
...
else:
```

Code to run if all previous logicals are false

13.3.1 Whitespace Like if . . . elif . . . else flowcontrol blocks, for loops are whitespace sensitive. The indentation of the line immediately below the for statement determines the indentation that all statements in the block must have. **13.3.2 break** A loop can be terminated early using break. break is usually used after an if statement to terminate the loop prematurely if some condition has been met.

```
x = randn(1000)
for i in x:
print(i)
if i > 2:
break
```

Since for loops iterate over an iterable with a fixed size, break is generally more useful in while loops.

13.3.3 continue continue can be used to skip an iteration of a loop, immediately returning to the top of the loop using the next item in iterable. continue is commonly used to avoid a level of nesting, such as in the following two examples.

```
x = randn(10)
for i in x:
if i < 0:
print(i)
for i in x:
if i >= 0:
continue
print(i)
```

Avoiding excessive levels of indentation is essential in Python programming – 4 is usually considered the maximum reasonable level. continue is particularly useful since it can be used to in a for loop to avoid one level of indentation.

1.2 break

break can be used in a while loop to immediately terminate execution. Normally, break should not be used in a while loop – instead the logical condition should be set to False to terminate the loop. However, break can be used to avoid running code below the break statement even if the logical condition is False.

```
condition = True
i = 0
x = randn(1000000)
while condition:
if x[i] > 3.0:
break # No printing if x[i] > 3
print(x[i])
i += 1
```

It is better to update the logical statement which determines whether the while loop should execute

```
i = 0
while x[i] <= 3:
print(x[i])
i += 1
```

1.2.1 13.4.2 continue

continue can be used in a while loop to skip any remaining code in the loop, immediately returning to the top of the loop, which then checks the while condition, and executes the loop if it still true. Using continue when the logical condition in the while loop is False is the same as using break.