COMP 10280
Programming I (Conversion)

John Dunnion

School of Computer Science
University College Dublin

COMP 10280 Programming I (Conversion)/Lecture 13

Outline

The `break` statement

- The `break` statement is used to break out of the closest enclosing `while` or `for` loop
- Like the `break` statement in C
- Control jumps to the statement immediately following the body of the statement containing the `break`
- If you want to exit from more than one loop, the `break` statement is not the thing to use
- "Of dubious legitimacy".
- Justifiable when an exceptional circumstance has occurred and the loop has to be abandoned

Program demonstrating the break statement in a for loop (1)

```
# Program to illustrate the use of the break statement in a loop
# Use in a for loop

for i in range(1000):
# Exit from the loop if the user enters 'quit' or 'exit'
    command = input('Enter a command:  ')
    if command == 'quit' or command == 'exit':
        break;
    print('i is:', i)

print('Finished!')
```

Program demonstrating the `break` statement in a `for` loop (2)

```
Enter a command:  create
i is:  0
Enter a command:  list
i is:  1
Enter a command:  help
i is:  2
Enter a command:  exit
Finished!
```

The `continue` statement

- The `continue` statement is used to start the next iteration of the closest enclosing `while` or `for` loop
- Like the `continue` statement in C
- Control jumps to the first statement of the body of the statement containing the `continue` (or to the next statement after the loop if there are no more iterations)
- Usually part of a conditional statement within the loop

Program demonstrating the continue statement in a for loop (1)

```
# Program to illustrate the use of the continue statement in a loop
# Use in a for loop

for num in range(2, 20):
# Continue with the next iteration of the loop
# if the number is divisible by 3
    if num % 3 == 0:
        print('Found a number divisible by 3:', num)
        continue
    print('Found a number', num)

print('Finished!')
```

Program demonstrating the `continue` statement in a `for` loop (2)

```
Found a number 2
Found a number divisible by 3:  3
Found a number 4
Found a number 5
Found a number divisible by 3:  6
Found a number 7
Found a number 8
Found a number divisible by 3:  9
Found a number 10
Found a number 11
Found a number divisible by 3:  12
Found a number 13
Found a number 14
Found a number divisible by 3:  15
Found a number 16
Found a number 17
Found a number divisible by 3:  18
Found a number 19
Finished!
```

Use of the `continue` statement

- The `continue` statement could be replaced by having the code after the `continue` inside a conditional statement
- This would require having an extra level of indentation, etc for that code
- Makes the code more cluttered, and harder to read(?)
- Used by Python programmers (just as C and Java programmers use the `continue` statement in those languages)

The `pass` statement

- The `pass` statement in Python does nothing(!!)
- The `pass` statement is a null operation; nothing happens when it executes
- It can be used when a statement is required syntactically but you do not want or need any command or code to be executed
- The `pass` statement is also useful in places where your code will eventually go but has not been written yet, eg in stubs
- It is also helpful when you have created a code block but it is no longer required
- You can then remove the statements inside the block but let the block remain with a `pass` statement so that it doesn't interfere with other parts of the code

Program demonstrating the pass statement in a for loop (1)

```
# Program to illustrate the use of the pass statement
# Used in a for loop

for letter in 'Python program':
    if letter == 'o':
        pass
        print('This is inside the pass block')
    print('Current Letter :', letter)

print('Finished!')
```

Program demonstrating the pass statement in a `for` loop (2)

```
Current Letter :  P
Current Letter :  y
Current Letter :  t
Current Letter :  h
This is inside the pass block
Current Letter :  o
Current Letter :  n
Current Letter :
Current Letter :  p
Current Letter :  r
This is inside the pass block
Current Letter :  o
Current Letter :  g
Current Letter :  r
Current Letter :  a
Current Letter :  m
Finished!
```

Other uses of the `pass` statement (1)

- Another place the `pass` statement can be used is as a place-holder for a function or conditional body when the programmer is working on new code
- This allows the programmer to keep thinking at a more abstract level
- The `pass` statement is silently ignored

```
while command != 'exit' and command != 'quit':
    if command == 'create':
        pass # Don't forget to implement this!

def my_new_function:
    pass # Don't forget to implement this!
```

Other uses of the `pass` statement (2)

- The `pass` statement is also commonly used for creating minimal classes

```
class MyNewClass:
    pass
```

`else` clause on loop statements

- Loop statements may have an else clause
- The `else` clause is executed when the condition becomes false (with the `while` loop) or when the loop terminates through exhaustion of the list (with the `for` loop)
- The `else` clause is not executed when the loop is terminated by a `break` statement

Program demonstrating the else clause in a for loop (1)

```
# Program to illustrate the use of the else statement on a for loop
# Search for prime numbers in a range of integers

# Look for prime numbers in a range of integers
for number in range(2, 20):
    for i in range(2, number):
        if number % i == 0:
            print(number, 'equals', i, '*', number//i)
            break
    else:
        # loop fell through without finding a factor
        print(number, 'is a prime number')

print('Finished!')
```

Program demonstrating the `else` clause in a `for` loop (2)

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
Finished!
```

Exiting a Python program

- Many languages provide a mechanism to exit a program
- Python provides four functions: `exit()`, `quit()`, `sys.exit()` and `os.exit()`
- `exit()` and `quit()` and aliases for one another. They should only be used in the shell
- `sys.exit()` raises the `SystemExit` exception in the background
- This means that it is the same as `exit()` and `quit()` in that respect
- However, unlike `exit()` and `quit()`, it is considered good practice to use `sys.exit()` in production code
- `os.exit()` exits the program without calling cleanup handlers, flushing stdio buffers, etc
- Thus, it is not a standard way to exit and should only be used in special cases

Program demonstrating the use of the `sys.exit()` function (1)

```
# Program to illustrate the use of the sys.exit() function

import sys

while True:
    answer = input('Do you want to continue?:  ')
    if answer == 'yes':
        print('OK, carry on then.')
    elif answer == 'no':
        print('Ciao!')
        sys.exit()

print('Finished!')
```

Program demonstrating the use of the `sys.exit()` function (2)

```
Do you want to continue?:  yes
OK, carry on then.
Do you want to continue?:  yes, please
Do you want to continue?:  I think so
Do you want to continue?:  no
Ciao!
```