MATH6005 Lecture 2

January 22, 2019

2 Conditionals and Loops

2.1 Topics covered

- Boolean operations
- Conditional logic
- Introduction to loops

2.1.1 Boolean Operators

- We have seen that Python has a type `bool` that can take two values `True` or `False`
- Boolean operators return `True` and `False` (or yes/no) answers

- **Tip**: Remember that the = operator is for **assignment**
- use == when you want to **compare** equality

```
In [2]: age = 25
        print(age < 40)
        print(age >= 40)
        print(age == 25)
```

```
True
False
True
```

2.2 Conditional Logic

- Most code makes extensive use of **if then** statements
- This makes use of the `if`, `elif`, and `else` keywords

Operation	Description	Operation	Description
a == b	a equal to b	a != b	a not equal to b
a <b	a less then b	a >b	a greater than b
a <= b	a less than or equal to b	a >= b	a greater than or equal to b

```
if <Boolean Operation 1>:
    do something
elif <Boolean Operation 2>:
    do something different
else:
    take default action
```

Example 1: Is the number less than 2?

```
In [3]: number = 5

        if number > 2:
            print("Number > 2!")
        else:
            print("Number <=2")
```

```
Number > 2!
```

2.2.1 Python whitespace

- The indenting in the previous example is mandatory in Python
- If you do not indent your `if then` statements then Python will throw an `IndentationError` exception

```
In [52]: number = 1
         if number > 2:
         print("Number > 2!")
         else:
         print("Number <=2")
```

```
          File "<ipython-input-52-44d8c9689021>", line 3
        print("Number > 2!")
            ^
    IndentationError: expected an indented block
```

- A more complex **if then**

```
In [7]: lower_limit = 10
        upper_limit = 15

        number_to_check = 12

        if number_to_check >= lower_limit and number_to_check <= upper_limit:
            print('number is inside range')
        else:
            print('number is outside of range')
```

```
number is inside range
```

- An example using `elif`

```
In [8]: age = 32
        pensionable_age = 68

        if age < 0:
            print('Error. Please enter an age greater than zero')
        elif age < pensionable_age:
            years_to_pension = pensionable_age - age
            print('There are {0} years until you can draw your pension!'.format(years_to_pension
        else:
            print('You are eligible to draw your_pension')
```

```
There are 36 years until you can draw your pension!
```

2.3 Functions and if statements

- we could also use a function within an `if` statement

```
In [1]: def eligible_for_pension(age):
            '''
            Return boolean True or False to indicate if
            person is eligible for their pension

            Keyword arguments:
            age -- a persons age in integer years (e.g. 32 or 45)
            '''
            return age >= 68
```

```
In [2]: age = 68

        if eligible_for_pension(age):
            print('Congratulations you can retire!')
        else:
            print('You are not yet eligible for your pension')
```

```
Congratulations you can retire!
```

- We can also use the `not` operator in Python

```
In [ ]: age = 68

        if not eligible_for_pension(age):
            print('You are not yet eligible for your pension')
        else:
            print('Congratulations you can retire!')
```

2.4 Nested if statements

```
In [9]: def stamp_duty(house_price, first_time_buyer):
            '''
            First time buyers recieve more tax relief
            than people buying their next home.
            Returns float representing the stamp duty owed.
            '''
            if first_time_buyer:
                #this if statement is nested within the first
                if house_price <= 300000:
                    return 0.0
                else:
                    return house_price * 0.05
            else:
                if house_price < 125000:
                    return 0.0
                else:
                    return house_price * 0.05
```

```
In [5]: my_first_house_price = 310000
        owed = stamp_duty(my_first_house_price, True)
        print('stamp duty owed = £{0}'.format(owed))
```

```
stamp duty owed = £15500.0
```

2.5 Introduction to iterating over data using loops

Algorithms often need to do the same thing again and again

For example, an algorithm making three servings of toast
Making Toast Algorithm:

1. Put a slice of bread in the toaster
2. Push lever down to turn on the toaster
3. Remove the toasted bread from the toaster
4. Put a slice of bread in the toaster
5. Push lever down to turn on the toaster
6. Remove the toasted bread from the toaster
7. Put a slice of bread in the toaster
8. Push lever down to turn on the toaster
9. Remove the toasted bread from the toaster

A better way to do this in code is to use a LOOP

Do the following 3 steps, 3 times: 1. Put a slice of bread in the toaster 2. Push lever down to turn on the toaster 3. Remove the toasted bread from the toaster

- There are two types of loop in Python
- `for` loops and `while` loops
- We generally use `while` if we do not know the number of iterations in advance
- We generally use `for` if we know the number of iterations in advance

2.5.1 While loops

```
In [2]: age = 15

        while age <= 18:

            print('you are currently {0} years old'.format(age))

            age += 1
```

```
you are currently 15 years old
you are currently 16 years old
you are currently 17 years old
you are currently 18 years old
```

2.5.2 While Loop Structure

- All while loops have the same structure
- You use the `while` keyword
- Followed by a boolean operation (e.g. `age <= 18`)
- Beware of infinite loops!
- Let's test a more complex while loop using a function

```
In [10]: def fizzbuzz(n):
             """
             For multiples of three print "Fizz" instead of the number
             and for the multiples of five print "Buzz".
             For numbers which are multiples of both three
             and five print "FizzBuzz".

             Keyword arguments:
             n -- the number to test
             """
             if n % 3 == 0 and n % 5 == 0:
                 print('fizzbuzz')
             elif n % 3 == 0:
                 print('fizz')
             elif n % 5 == 0:
                 print('buzz')
             else:
                 print(n)
```

```
In [12]: n = 1
         limit = 15

         whi le n <= limit:
             fizzbuzz(n)
             n += 1
```

```
1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
11
fizz
13
14
fizzbuzz
```

- A `while` loop example where we do **not** know the number of iterations in advance.

```
In [18]: list_to_search = ['we', 'are', 'the', 'knights', 'who', 'say', 'ni']
         not_found = True
         word_to_find = 'knights'
         current_index = 0
         index_of_word = -1

         while not_found:
             if list_to_search[current_index] == word_to_find:
                 not_found = False
                 index_of_word = current_index
             current_index += 1

         print("the word '{0}' is located in index {1}".format(word_to_find, index_of_word))
```

```
the word 'knights' is located in element 3
```

- The previous example can easily lead to an `IndexError`.
- If `word_to_find` was 'shrubbery' then the loop would exhaust all list elements
- Although we do not know the number of iterations needed, we can easily modify the `while` loop to take account of the maximum allowable iterations.

```
In [20]: list_to_search = ['we', 'are', 'the', 'knights', 'who', 'say', 'ni']
         not_found = True
         word_to_find = 'shrubbery'
         current_index = 0
         index_of_word = -1
         word_count = len(list_to_search)

         while not_found and current_index < word_count:
```

```
            if list_to_search[current_index] == word_to_find:
                not_found = False
                index_of_word = current_index
            current_index += 1

        if not_found:
            print("'{0}' could not be found".format(word_to_find))
        else:
            print("'{0}' is located in index {1}".format(word_to_find, index_of_word))
```

```
'shrubbery' could not be found
```

2.5.3 For loops

- To create a **for** loop you need the following:
- `for` keyword
- a variable
- the `in` keyword
- the `range()` function - which is a built-in function in the Python library to create a sequence of numbers

```
In [49]: for age in range(5):
             print('you are currently {0} years old'.format(age) )
```

```
you are currently 0 years old
you are currently 1 years old
you are currently 2 years old
you are currently 3 years old
you are currently 4 years old
```

- notice that the loop sets age to 0 initially!
- `range()` takes keyword arguments to set the start (inclusive, default = 0), end (exclusive) and step

```
In [50]: for age in range(1, 5):
             print('you are currently {0} years old'.format(age))
```

```
you are currently 1 years old
you are currently 2 years old
you are currently 3 years old
you are currently 4 years old
```

```
In [54]: for age in range(1, 5, 2):
             print('you are currently {0} years old'.format(age))
```

```
you are currently 1 years old
you are currently 3 years old
```

```
In [56]: limit = 15

         for n in range(1, limit+1):
             fizzbuzz(n)
```

```
1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
11
fizz
13
14
fizzbuzz
```

2.5.4 Watch out for python whitespace rules!

- Remember to indent the next line after :

```
In [136]: limit = 15

          for n in range(1, limit+1):
          fizzbuzz(n)


          File "<ipython-input-136-a9a12e790b42>", line 4
        fizzbuzz(n)
               ^
    IndentationError: expected an indented block
```

2.6 Nested Loops

- `for` and `while` loops can be nested within each other.
- Think of nested loops as a 'loop of loops'
- Remember that for each iteration outer loop will consist of multiple iterations of an inner loop

- **Don't panic** if you do not understand straight away!

```
In [13]: for outer_index in range(3):
             print('Outer loop iteration: {0}'.format(outer_index))

             for inner_index in range(5):
                 print('\tInner loop iteration: {0}'.format(inner_index))
```

```
Outer loop iteration: 0
        Inner loop iteration: 0
        Inner loop iteration: 1
        Inner loop iteration: 2
        Inner loop iteration: 3
        Inner loop iteration: 4
Outer loop iteration: 1
        Inner loop iteration: 0
        Inner loop iteration: 1
        Inner loop iteration: 2
        Inner loop iteration: 3
        Inner loop iteration: 4
Outer loop iteration: 2
        Inner loop iteration: 0
        Inner loop iteration: 1
        Inner loop iteration: 2
        Inner loop iteration: 3
        Inner loop iteration: 4
```

- Example 2: The inner loop now iteratures **backwards**

```
In [14]: for outer_index in range(2):
             print('Outer loop iteration: {0}'.format(outer_index))

             for inner_index in range(5, 0, -1):
                 print('\tInner loop iteration: {0}'.format(inner_index))
```

```
Outer loop iteration: 0
        Inner loop iteration: 5
        Inner loop iteration: 4
        Inner loop iteration: 3
        Inner loop iteration: 2
        Inner loop iteration: 1
Outer loop iteration: 1
        Inner loop iteration: 5
        Inner loop iteration: 4
        Inner loop iteration: 3
        Inner loop iteration: 2
        Inner loop iteration: 1
```

Nested Loops Example 2. Let's use a nested `for` loop to create the pattern below

```
1
12
123
1234
12345
```

```
In [33]: for outer_index in range(1, 6):
             #remember this is a loop of loops.
             #the loop below execute all iterations each
             # time the outer loop iterates
             for inner_index in range(1, outer_index + 1):

                 #we use the end='' option of print so
                 #that it prints on the same line as previous
                 print(inner_index, end='')

             #new line
             print('')
```

```
1
12
123
1234
12345
```

2.7 Lab work

- Lab 2 is now available on blackboard.
- It explores conditionals and loops.
- Please take a look at lab 2 before you come along this week.
- Please ask us questions in the lab if you do not understand something.