M30299 – Programming Lecture 11 – Using If Statements & For Loops

Matthew Poole & Nadim Bakhshov moodle.port.ac.uk

School of Computing University of Portsmouth

2020/21

Introduction to lecture

- We begin this lecture with some more examples of if statements.
- We then turn our attention to Python's for loop.
- Consider a pizza restaurant that uses the following function to calculate the price of its pizzas:

```
def priceOfPizza(diameter):
    area = math.pi * (diameter / 2) ** 2
    pricePerSquareCm = 0.015
    return pricePerSquareCm * area
```

• The restaurant is not making enough money on small pizzas and decides to charge £2 extra for pizzas smaller than 400cm².

Designing decision structures

• One way to re-write the function is by using an if-else statement, as follows:

```
def priceOfPizza(diameter):
    area = math.pi * (diameter / 2) ** 2
    pricePerSquareCm = 0.015
    if area < 400:
        return pricePerSquareCm * area + 2
    else:
        return pricePerSquareCm * area</pre>
```

- This function is correct, and seems reasonably readable.
- What might be the main criticism?

Designing decision structures

- A better solution may be to calculate the "basic" price of a pizza, and then add 2 to this in appropriate cases.
- For this, we use an extra variable and a simpler if statement:

```
def priceOfPizza(diameter):
    area = math.pi * (diameter / 2) ** 2
    pricePerSquareCm = 0.015
    price = pricePerSquareCm * area
    if area < 400:
        price = price + 2
    return price</pre>
```

Decision structures: a final example

• The following priceOfPizza function allows for the fact that ingredients of different pizzas are more expensive than others:

```
def priceOfPizza(diameter, flavour):
    if flavour == "supreme feast":
        pricePerSquareCm = 0.018
    elif flavour == "cheese and tomato":
        pricePerSquareCm = 0.012
    else:
        pricePerSquareCm = 0.015
    area = math.pi * (diameter / 2) ** 2
    return area * pricePerSquareCm
```

for loops - a review

- We have already used some for loops in the practicals.
- For loops are used to loop or iterate through a sequence of values, such as a list or a string:

• A for loop includes a **loop variable**, a **sequence** and **body**.

The range function

• Most for loops use the built-in function range:

```
>>> range(5)
range(0, 5)
>>> type(range(5))
<class 'range'>
```

- We see that the range function just gives us an object of type range.
- A range object **represents** a **sequence** of values.

The range function

 In order to see what these values are we need to generate a list from the range object:

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

- So, range(n) gives a sequence of length n that begins with 0.
- This sometimes leads to awkward arithmetic in the loop body:

```
def countToFive():
    for i in range(5):
        print(i + 1)
```

for loops - using range

• We can avoid this problem by using two arguments to range:

```
>>> list(range(1, 6))
[1, 2, 3, 4, 5]
>>> list(range(10, 20))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

 We see that range(m, n) gives a sequence starting with m and finishing one short of n. So, we can now write:

```
def countToFive():
    for i in range(1, 6):
        print(i)
```

for loops - using range

• We can also use range with three arguments:

```
>>> list(range(0, 10, 2))
[0, 2, 4, 6, 8]
>>> list(range(10, 0, -1))
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

• We see that range (m, n, s) gives a sequence that starts with m, steps every s, and stops just short of n. For example:

```
def countDownFromFive():
   for i in range(5, 0, -1):
      print(i)
```

• Like all control structures, we can **nest** one for loop within another:

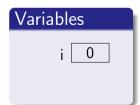
```
Code
  for i in range(3):
    for j in range(2):
       print("i =", i, "j =", j)
    print("========")
```

Variables



• Like all control structures, we can **nest** one for loop within another:

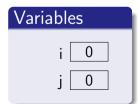
```
Code
> for i in range(3):
    for j in range(2):
        print("i =", i, "j =", j)
    print("========")
```





• Like all control structures, we can **nest** one for loop within another:

```
Code
  for i in range(3):
>    for j in range(2):
        print("i =", i, "j =", j)
        print("========")
```





• Like all control structures, we can **nest** one for loop within another:

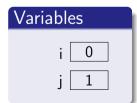
```
Code
  for i in range(3):
      for j in range(2):
>      print("i =", i, "j =", j)
      print("========")
```

Variables i 0 j 0

Screen $i = 0 \ j = 0$

• Like all control structures, we can **nest** one for loop within another:

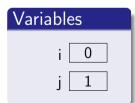
```
Code
  for i in range(3):
>    for j in range(2):
        print("i =", i, "j =", j)
        print("========")
```



Screen $i = 0 \ j = 0$

• Like all control structures, we can **nest** one for loop within another:

Code for i in range(3): for j in range(2): > print("i =", i, "j =", j) print("========")



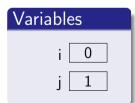
Screen

$$i = 0 j = 0$$

 $i = 0 j = 1$

• Like all control structures, we can **nest** one for loop within another:

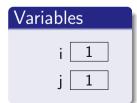
```
Code
  for i in range(3):
     for j in range(2):
        print("i =", i, "j =", j)
> print("=========")
```



Screen i = 0 j = 0i = 0 j = 1

• Like all control structures, we can **nest** one for loop within another:

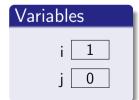
Code > for i in range(3): for j in range(2): print("i =", i, "j =", j) print("========")

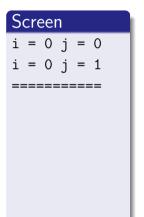


Screen i = 0 j = 0i = 0 j = 1

• Like all control structures, we can **nest** one for loop within another:

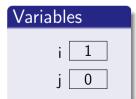
```
Code
  for i in range(3):
>    for j in range(2):
        print("i =", i, "j =", j)
        print("========")
```





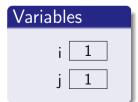
• Like all control structures, we can **nest** one for loop within another:

```
Code
  for i in range(3):
      for j in range(2):
>         print("i =", i, "j =", j)
      print("========")
```



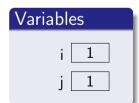
• Like all control structures, we can **nest** one for loop within another:

```
Code
  for i in range(3):
>    for j in range(2):
        print("i =", i, "j =", j)
        print("========")
```



• Like all control structures, we can **nest** one for loop within another:

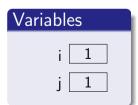
```
Code
  for i in range(3):
        for j in range(2):
        print("i =", i, "j =", j)
        print("========")
```



Screen i = 0 j = 0i = 0 i = 1i = 1 j = 0i = 1 i = 1

• Like all control structures, we can **nest** one for loop within another:

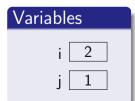
```
Code
  for i in range(3):
     for j in range(2):
        print("i =", i, "j =", j)
> print("=========")
```

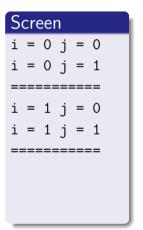


Screen $i = 0 \ j = 0$ i = 0 i = 1i = 1 i = 0i = 1 i = 1

• Like all control structures, we can **nest** one for loop within another:

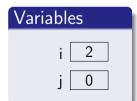
```
Code
> for i in range(3):
    for j in range(2):
        print("i =", i, "j =", j)
    print("========")
```

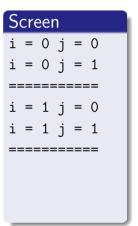




• Like all control structures, we can **nest** one for loop within another:

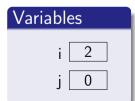
```
Code
  for i in range(3):
>    for j in range(2):
        print("i =", i, "j =", j)
        print("========")
```





• Like all control structures, we can **nest** one for loop within another:

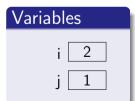
```
Code
  for i in range(3):
        for j in range(2):
        print("i =", i, "j =", j)
        print("========")
```



Screen $i = 0 \ j = 0$ i = 0 j = 1i = 1 i = 0i = 1 i = 1i = 2 j = 0

• Like all control structures, we can **nest** one for loop within another:

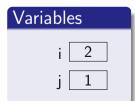
```
Code
  for i in range(3):
>    for j in range(2):
        print("i =", i, "j =", j)
        print("========")
```



Screen $i = 0 \ j = 0$ i = 0 j = 1i = 1 i = 0i = 1 i = 1i = 2 j = 0

• Like all control structures, we can **nest** one for loop within another:

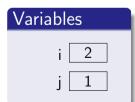
```
Code
  for i in range(3):
        for j in range(2):
        print("i =", i, "j =", j)
        print("========")
```

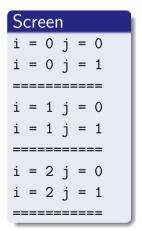


Screen $i = 0 \ j = 0$ i = 0 i = 1i = 1 j = 0i = 1 i = 1i = 2 j = 0i = 2 j = 1

• Like all control structures, we can **nest** one for loop within another:

```
Code
  for i in range(3):
     for j in range(2):
        print("i =", i, "j =", j)
>     print("=========")
```





- Let's consider a problem that requires the use of for loops.
- We'll write a timesTable function that has a parameter n and displays a "times table" for all numbers up to n.
- For example, timesTable(4) would display:
 - 4 8 12 16
 - 3 6 9 12
 - 2 4 6 8
 - 1 2 3 4

- We can see that:
 - The first line contains the n-times table;
 - The second line contains the (n-1)-times table; ...
 - The n-th line contains the 1-times table.
- This is clearly the job of a for loop, and we can use range(n, 0, −1) to give the numbers n downto 1
- A first stab at an algorithm to solve our problem is therefore:

```
for i in range(n, 0, -1):

display the i-times table on a line
```

• We can convince ourselves that this partially complete code works by converting the English into a print statement ...

• For example, we can write the following:

```
def timesTable(n):
   for i in range(n, 0, -1):
      print(i, "times table")
```

and try calling timesTable(4).

- Now, we are left with a smaller problem: displaying the i-th times table on a line (to replace the above print statement).
- This involves displaying the values $i \times 1, i \times 2, \dots, i \times n$ on a line...
- ... we thus need to multiply i with every number from 1 to n.

- Again, this is a job for a for loop, we can use range(1, n + 1) to give a sequence
 of appropriate numbers.
- All the numbers should be printed on the same line (i.e. no newlines between them), but we need a newline at the end.
- This is achieved in Python using the following code:

```
for j in range(1, n + 1):
    print(i * j, end=" ")
print()
```

- A final consideration is to make sure the numbers in the table line up well (some numbers have more digits that others).
- This is achieved using the string format method; e.g.,

```
print("{0:3}".format(i * j), end="")
```

allocates three characters to each number in the table. So, finally:

```
def timesTable(n):
    for i in range(n, 0, -1):
        for j in range(1, n + 1):
            print("{0:3}".format(i * j), end="")
        print()
```