Algorithms and Repetition

Algorithms and repetition

- The **flow of control** is the order that statements are executed in a program.
- Unless otherwise specified, Python programs start at the first statement in the file, and finish with the last.
- This behaviour can be modified with **conditional** and **repetition** statements.
- Any algorithm can be expressed in terms of conditional and repetition statements.
- **Repetition statements** allow the same thing (or almost the same thing) to be done again and again until a particular **condition** is broken.
- The repetition statements in Python are the **while** statement and the **for** statement.
- In this video we will cover **for** loops.

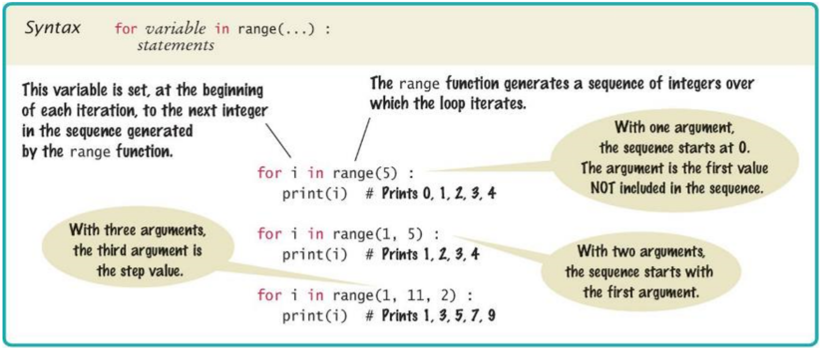
An example needing repetition

Try the Exchange Table example: src/exchange_table.py

```
uk_amount = float(input("\nEnter a starting amount in UK sterling: "))
print("\n\tPOUNDS\t\tEUROS")
print("\t{:.2f}\t\t{:.2f}".format(uk_amount,
                                  uk_amount * EXCHANGE_RATE))
uk_amount = uk_amount + 10
print("\t{:.2f}\t\t{:.2f}".format(uk_amount,
                                  uk_amount * EXCHANGE_RATE))
uk_amount = uk_amount + 10
print("\t{:.2f}\t\t{:.2f}".format(uk_amount,
                                  uk_amount * EXCHANGE_RATE))
```

The `for` loop with a range

You can use a `for` loop as a count-controlled loop to iterate over a range of integer values.



```
Syntax    for variable in range(...) :
              statements
```

```
for i in range(5) :
    print(i)   # Prints 0, 1, 2, 3, 4

for i in range(1, 5) :
    print(i)   # Prints 1, 2, 3, 4

for i in range(1, 11, 2) :
    print(i)   # Prints 1, 3, 5, 7, 9
```

This variable is set, at the beginning of each iteration, to the next integer in the sequence generated by the range function.
The range function generates a sequence of integers over which the loop iterates.
With one argument, the sequence starts at 0. The argument is the first value NOT included in the sequence.
With three arguments, the third argument is the step value.
With two arguments, the sequence starts with the first argument.

Figure 1: Diagram from *Python for Everyone*, Horstmann and Necaise

Example with a for statement

This version of the Exchange Table program which a `for` loop: src/exchange_table_with_for.py

```
NUM_LINES = 10
EXCHANGE_RATE = 1.564

print("\n\tPOUNDS\t\tEUROS")

uk_amount = 0
for line_counter in range(NUM_LINES):
    print("\t{:.2f}\t\t{:.2f}".format(uk_amount,
                                        uk_amount * EXCHANGE_RATE))
    uk_amount = uk_amount + 10
```

Example with nested for loops

This program prints out a series of times tables: src/times_tables.py

```
first_number = int(input("\n\tEnter the number of the first
                          times table you want: "))

last_number = int(input("\tEnter the number of the last
                         times table you want: "))

total_lines = int(input("\tEnter the number of lines for each
                         table: "))

for table_number in range(first_number, last_number + 1):
    print("")

    for line_number in range(1, total_lines + 1):
        print("\t\t{} x {} = {} ".format(table_number,
                    line_number, (table_number * line_number)))
```

Algorithms and problem solving

- Each programming task needs its own algorithm, possibly with (nested) conditional and repetition statements.
- There are no “magic formulas” or automatic recipes for designing algorithms.
- Designing an algorithm is a **problem solving** activity.
- **Understanding** and **analysing** the problem or task is an essential first step.
- **Pseudocode** (i.e. half computer code, half natural language), and **diagrams** can help.
- Programming by trial and error **without understanding**, and programming **in a hurry**, nearly always **result in a mess**!

Questions

Question 1

What do these loops print?

```
a. for i in range(1, 10) :
      print(i)

b. for i in range(1, 10, 2) :
      print(i)

c. for i in range(10, 1, -1) :
      print(i)

d. for i in range(10) :
      print(i)

e. for i in range(1, 10) :
      if i % 2 == 0 :
         print(i)
```

(Horstmann, p 226)

Horstmann, Cay S., Rance Necaise. Python for Everyone, Interactive Edition, 2nd Edition. Wiley, 2016-05-09. VitalBook file.

Answer 1

a.
```
for i in range(1, 10) :
    print(i)
```

The output is: 1, 2, 3, 4, 5, 6, 7, 8, 9

b.
```
for i in range(1, 10, 2) :
    print(i)
```

The output is: 1, 3, 5, 7, 9

c.
```
for i in range(10, 1, -1) :
    print(i)
```

The output is: 10, 9, 8, 7, 6, 5, 4, 3, 2

Answer 1 (cont.)

```
d. for i in range(10) :
       print(i)
```

The output is: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

```
e. for i in range(1, 10) :
       if i % 2 == 0 :
          print(i)
```

The output is: 2, 4, 6, 8

Question 2

What is an “off-by-one” error? Give an example from your own programming experience.

(Horstmann p226) Horstmann, Cay S., Rance Necaise. Python for Everyone, Interactive Edition, 2nd Edition. Wiley, 2016-05-09. VitalBook file.

Answer 2

What is an "off-by-one" error? Give an example from your own programming experience.

Answer An "off-by-one" error refers to a loop that runs one time too many or one time too few. One common source of off-by-one errors is forgetting that the values generated by range are asymmetric (the second bound is not included in the range), and as a result, the loop runs one time too few.