# Repeating Things
# Chapters 2.6-2.7

Photo from RCReptiles.com

# Learning Outcomes

- To understand the concept of repetitive flow control.
- To learn how to use simple loops to repeat program statements.
- To understand the concept of ranges and how they are applied to programming in Python.

# FLOW CONTROL

- Flow: The order in which program statements (or algorithm steps) are executed.
- Flow Control: How to control the order in which program statements are executed.
- Flow control is very important for the correctness of the program!
- Take our currency conversion algorithm
  - Step 1: Get the amount in RMB.
  - Step 2: Calculate the amount in HKD.
  - Step 3: Display the calculated amount to the user.
- We could do Step 2, then Step 1, then Step 3 -- but that wouldn't give us the correct answer!

# Sequential Flow Control

- Ordinarily, the computer assumes that program statements are to be executed **sequentially**.
- This means that the program statement at the top of the function is executed first, then the next one, etc -- **from top to bottom, one at a time**.
- This is called **sequential flow**.

# ITERATIVE FLOW

- Sometimes, we might want to execute certain programming statements more than once.
- For example: Write a program that will calculate the average quiz score of all the students in a class.
- You know that there are 5 students in the class.

# OUR ALGORITHM

- Write down, in English, the steps that are needed to calculate the average score for a class of 5 students.

# WHAT'S WRONG WITH THIS?

```
def main():
    print("This program calculates the average score of 5 students")
    totalScore = 0
    userInput = input("Score for student:")
    score = eval(userInput)
    totalScore = totalScore + score
    userInput = input("Score for student:")
    score = eval(userInput)
    totalScore = totalScore + score
    userInput = input("Score for student:")
    score = eval(userInput)
    totalScore = totalScore + score
    userInput = input("Score for student:")
    score = eval(userInput)
    totalScore = totalScore + score
    userInput = input("Score for student:")
    score = eval(userInput)
    totalScore = totalScore + score
    average = totalScore/5                  # Calculates the average
    print("The average is", average)

main()
```

# What was wrong

- Several things about the previous program make it very stupid:
  - It doesn't make sense for the programmer to type in identical statements many, many times! Humans are prone to error, and the computer should be able to do this sort of repetitive, mindless stuff.
  - If we were to change the number of students in the class, then we would have to modify the program by adding/subtracting program statements before we could use it!

# Loops

- In programming, if we want to repeat certain statements, we can put them into a **loop**.
- A loop will repeat whatever statements are in its **loop body** for a specified number of times, or until it is told to stop.
- Python has two kinds of loops. We will start with the **counted loop**.

# Starting with Loops

- Try the following in the Python interpreter:

```
for i in [1, 2, 3, 4, 5]:
    print(i)
```

```
for j in [2, 4, 6, 8, 10, 12]:
    print(j)
```

```
for myVariable in [3, 7, 1, 2, 8, 2]:
    print(myVariable)
```

- What does the `for ... in ...` statement do?

# THE FOR LOOP

- A counted loop is also called a **`for` loop**.
- Python `for` loops have the following syntax:

```
for <variable> in <sequence>:
    <body>
```

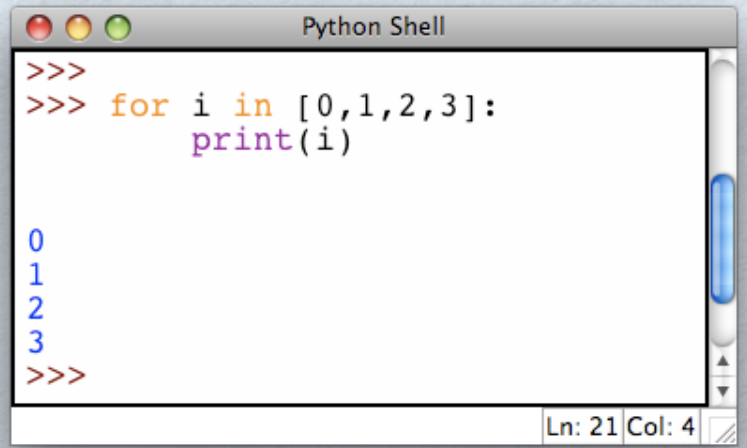
- The body of the loop is a sequence of program statements.
- The statements in the loop body are **indented** to indicate the start and the end of the loop.
- Python will perform the statements in the **loop body** once for every item in the sequence.

# THE FOR LOOP

```
for <variable> in <sequence>:
    <body>
```

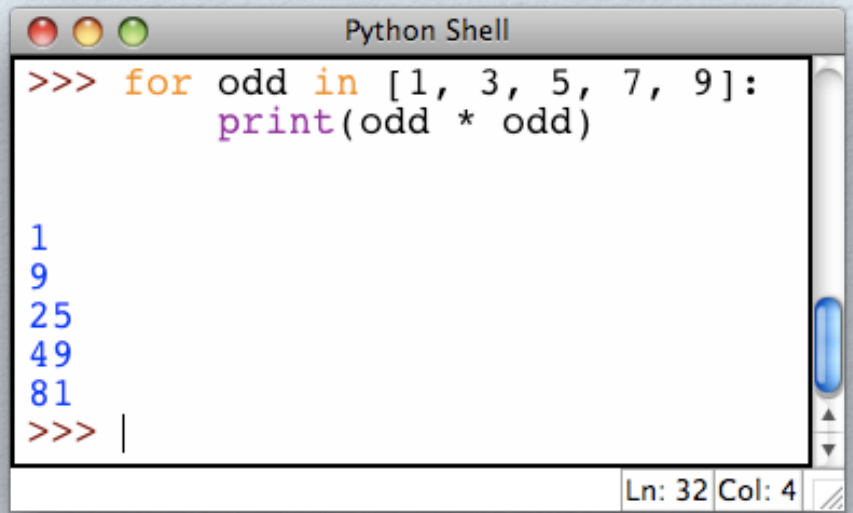
- The variable after the keyword `for` is called the **loop index**.
- The sequence is usually a list or a collection of values.
  - Python lists are a comma-separated collection of values that are enclosed in square brackets.
- It gets assigned each value in the sequence, and the statements in the loop body are performed once for each value.
- The loop stops when there are no more values in the sequence.

# AN EXAMPLE




- The variable `i` is the loop index.
- The sequence is made of the values `[0, 1, 2, 3]`
- The statement inside the loop body (`print()`) is executed once for each value in the sequence.
  - e.g. once for `i` = 0, once for `i` = 1, once for `i` = 2, once for `i` = 3.

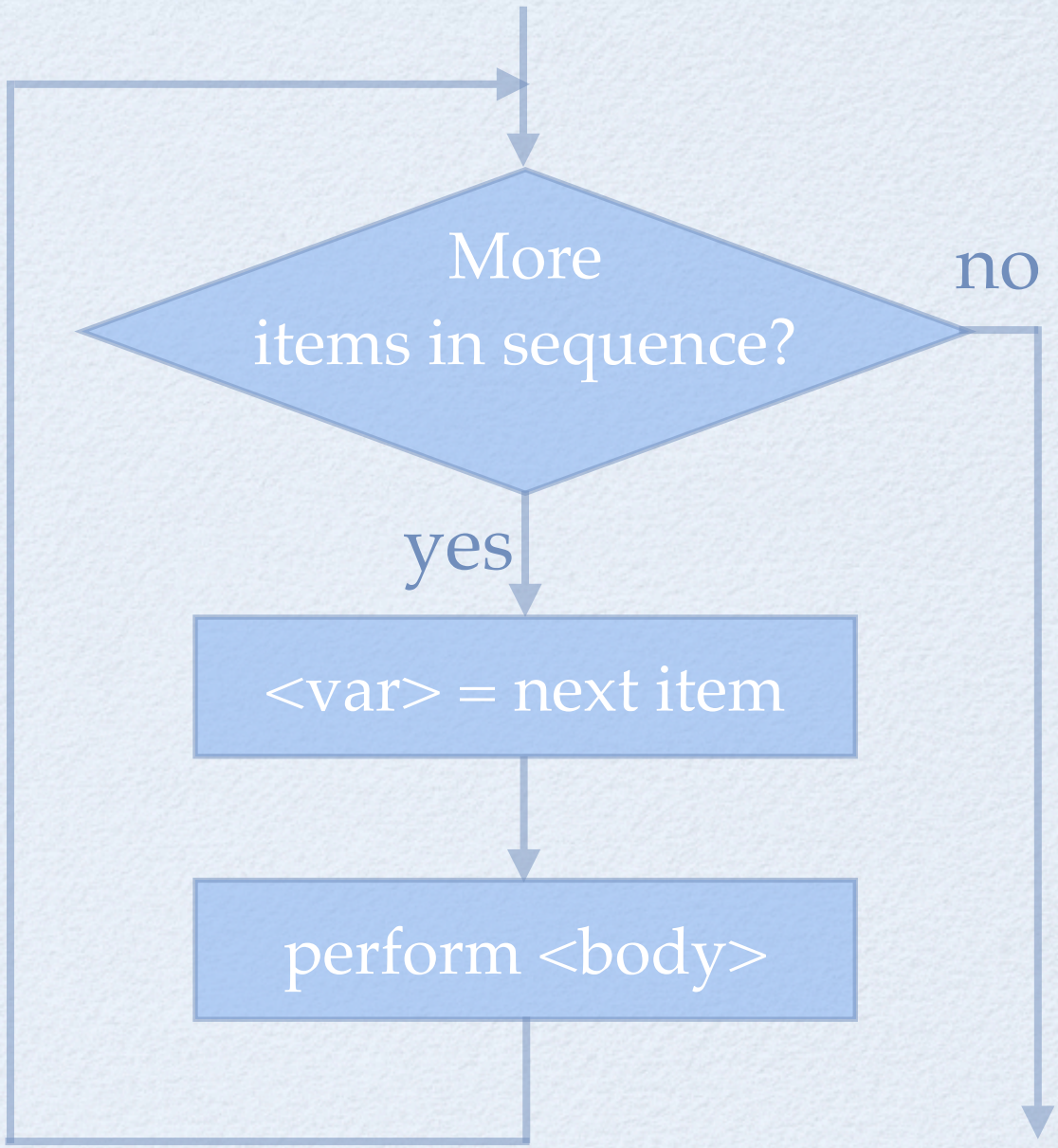
# ANOTHER EXAMPLE




- Circle the loop index.
- Circle the sequence.
- How many values are in the sequence? What are the values?
- Write down the values that are assigned to the loop index, in order.
- How many times does the loop execute?

# Loop Flowchart

- Flowcharts are a diagrammatic representation of the program flow (the order in which the program statements are executed).
- Diamonds: Decision points
- Rectangles: Statements



# EXERCISES

- Fill in the loop statement that is required to get the displayed output. Check your answer using the Python interpreter.

```
>>>
        print(i*2)
2
6
10
14
18
20
>>>
```

# EXERCISES

- Fill in the loop statement that is required to get the displayed output. Check your answer using the Python interpreter.

```
>>>
          print(i)
5
4
3
2
1
0
>>>
```

# EXERCISES

- Fill in the loop statement that is required to get the displayed output. Check your answer using the Python interpreter.

```
>>>
        print(i)
3
8
-1
35
22
0
>>>
```

# EXERCISES

- Fill in the loop statement that is required to get the displayed output. Check your answer using the Python interpreter.

```
>>>
        print(i)
8
8
8
8
8
>>>
```

# EXERCISES

- Using the program statements on the left, put them into the correct order with the correct indentation to get the program output on the right

```
for i in [1, 3, 5, 7, 9]:
print("total is", total)
total = 0
total = total + i
```

```
total is 36
```

# EXERCISES

- Using the program statements on the left, put them into the correct order with the correct indentation to get the program output on the right (note: not all of the program statements will get used!)

```
print("5! is", factorial)
factorial = 0
factorial = 1
for i in [1, 3, 5, 7, 9]:
for i in [1, 2, 3, 4, 5]:
for i in [2, 4, 3, 8, 1]:
factorial = factorial + i
factorial = factorial * i
```

```
5! is 120
```

# EXERCISE

- Rewrite the program on Slide 6 that calculates the average score from a class of 5 students.
- Your program should be called `averageScore.py` and must get the specified output.

```
Score for student? 55
Score for student? 80
Score for student? 90
Score for student? 10
Score for student? 87
Average score is 64.4
```

# FOR EXTRA CREDIT

- For extra credit, modify your `averageScore.py` program to get the new output below.
- Hints:
  - You cannot add a string and a number, but you can add two strings.
  - To turn a number into a string, use the `str()` function
    - `str(5)` will give you the string `"5"`
    - Try out your program statements in IDLE before writing your program.

```
Score for student 1? 55
Score for student 2? 80
Score for student 3? 90
Score for student 4? 10
Score for student 5? 87
Average score is 64.4
```

# More ways of making sequences

- Our way of creating sequences works for small numbers of iterations.
- However, suppose that we need to calculate the average score for the 120 students in COMP 201.
- There's got to be an easier way than writing [1, 2, 3, 4, ..., 119, 120].

# RANGES

- The `range()` function is very useful in `for` loops because it allows us to create sequences easily:

```
>>> for i in range(10):
    print(i,", ", end="")


0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 ,
```

- Python generates all the numbers from 0 to 10 (including 0 but not including 10), adding 1 every time.

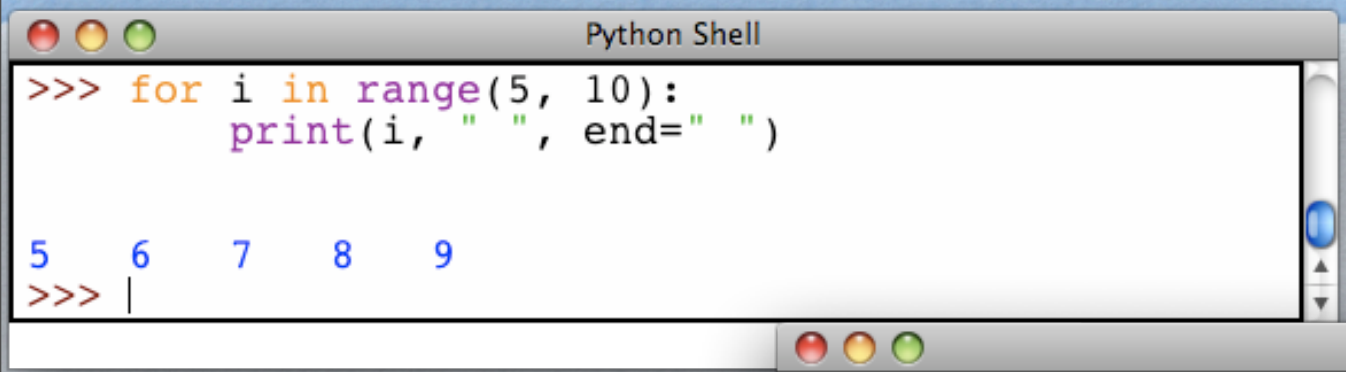
# RANGE()

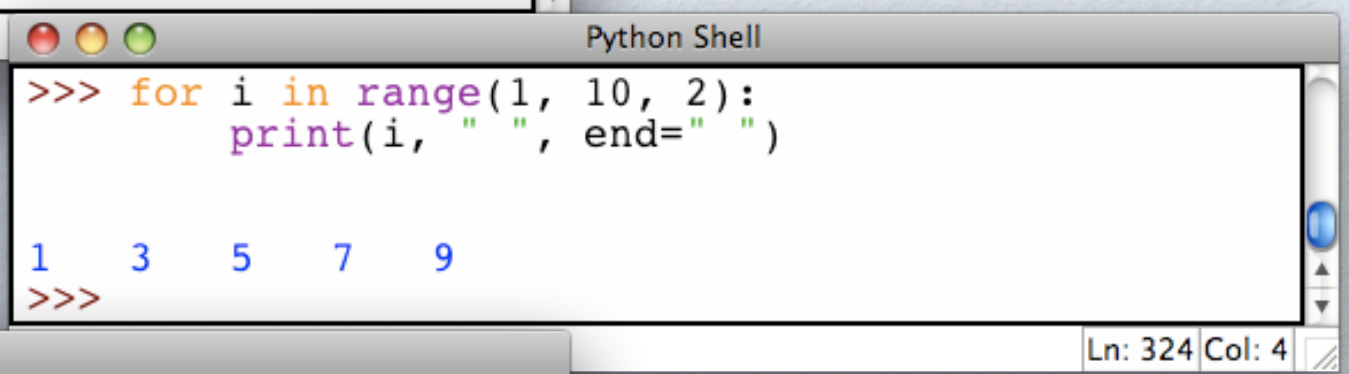
- The syntax of range() is:

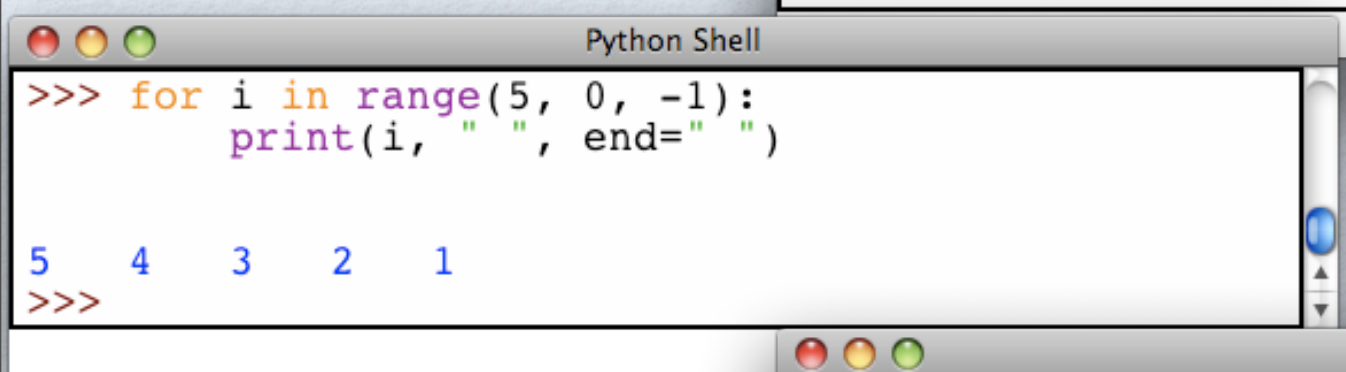
```
range([start, ]stop[, step])
```

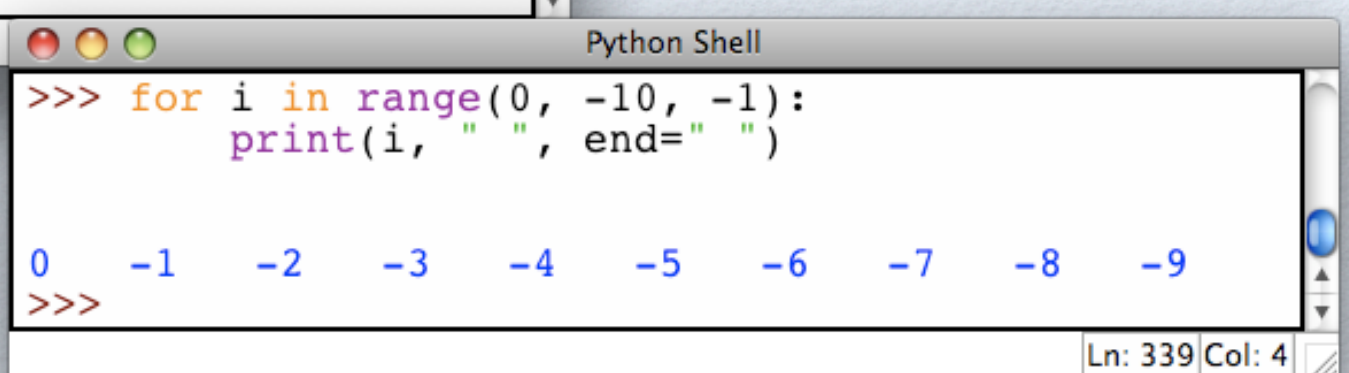
- Python generates all the numbers from *start* to stop (including *start* but not including stop), adding *step* every time
- *start* and *step* are optional.
- If we leave out *start*, then Python assumes that we want to start from 0.
- If we leave out *step*, then Python assumes that we want to add one every time.

# RANGE() EXAMPLES



# EXAMPLE

- Suppose we deposit $100 into a bank account that earns 3% compound interest per annum.
  - In other words, after one year, amount of the principal = principal * (1+0.03)
- Write a program, `FutureValue.py`, that would calculate the value of this investment after 10 years.
- Try to write down the inputs, output and the algorithm yourself first.

# EXAMPLE

- Inputs: The principal and the annual percentage rate, as a decimal number.
- Output: The value of the investment after 10 years.
- Algorithm:
  - Print an introduction
  - Input the amount of the principal
  - Input the annual percentage rate
  - Repeat 10 times:
    - Calculate principal = principal * (1+apr)
  - Output the value of principal

# EXERCISES

- Modify the Fahrenheit-to-Celsius converter program to print out a “table” of Fahrenheit temperatures and their Celsius equivalents every 10 degrees from 32F to 212F.

# EXERCISE

- Modify the FutureValue.py program so that the number of years is also a user input. (Be careful: the output statement at the end probably needs to be changed also!)

# You now know...

- What iterative flow control means.
- How to use definite loops in Python.
- How to choose and write appropriate ranges for your problems.