COMP1021
Introduction to Computer Science

Using For Loops

Gibson Lam and David Rossiter

Outcomes

- After completing this presentation, you are expected to be able to:
  1. Use the range command to generate a range of numbers
  2. Write loops using the for command

Writing Loops Using For

- Previously, we discussed the use of *while* loops to do things repeatedly in Python
- In this presentation we will look at another way of doing loops, using *for loops*
- Using a for loop:
  – you can perform some actions a particular number of times, or:
  – you can loop through a set of data, performing some actions on every item in the set

The Range Command

- Usually when you do for loops you use *range*
- The range command generates a range of numbers
- For example, you can generate the numbers 1 to 5 using the following code:

```
range(1, 6)
```

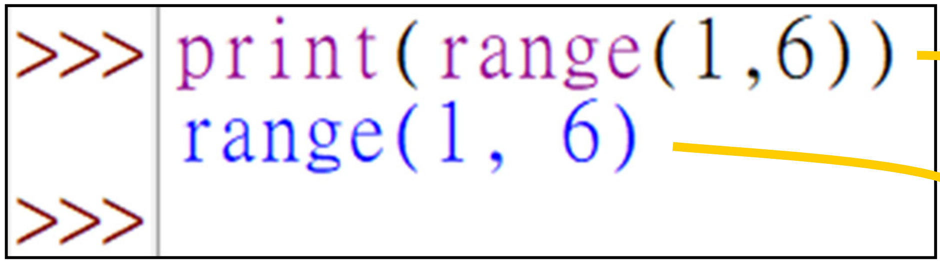
- The above line of code returns 1, 2, 3, 4 and 5

Showing a Range

- If you print the result of a range you will see an helpful result:

- *Your code*

- *Here we are using the shell, but you could do the same things in a program*

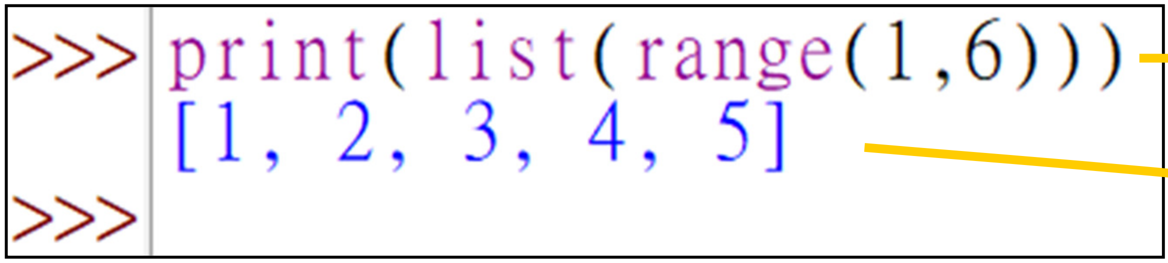


```
>>> print(range(1,6))
range(1, 6)
>>>
```

- *The result of your code*

- To see the numbers returned by the range command, use `list()`, like this:

- *Your code*



```
>>> print(list(range(1,6)))
[1, 2, 3, 4, 5]
>>>
```

- *The result of your code*

The Target Number

- If you look at this code:

```
range(1, 6)
```

- *The target number*

you may think that it should return the number 6 as well, because the range is from 1 to 6

- However, as you can see in previous slides the range command **does not** generate the target number
- In this example, the number 6 is not generated

The Starting Number

- If you do not provide the starting number the default starting value used by the range command will be 0 (not 1)
- For example, you can generate a range of numbers from 0 to 4 by using this line of code:

```
range(5)
```

- This code generates 0, 1, 2, 3 and 4 (again, the target number 5 is not generated by `range`)

The Step Value

- If you want, you can provide an optional third value in the range command
- The third value is called the *step* value
- You can use it to skip (=jump over) numbers
- For example, a step value of 2 will skip every other number, like this:

`range(1, 10, `**`2`**`)` returns 1, 3, 5, 7 and 9

`range(2, 10, `**`2`**`)` returns 2, 4, 6 and 8

Note that 10, the target number, is not included in the result

More Examples of the Range Command

- Here are more examples of using the step value:

  `range(0, 10, 3)` returns 0, 3, 6 and 9

  `range(-1,-10,-2)` returns -1, -3, -5, -7 and -9

- Here are some unusual examples of using `range()`:

  `range(10, 1)`

  returns nothing because the default step value is 1

  `range(-10, 1,-1)`

  returns nothing because the step value is -1

  `range(0, 10, 0)`

  results in an error because the step value must not be zero

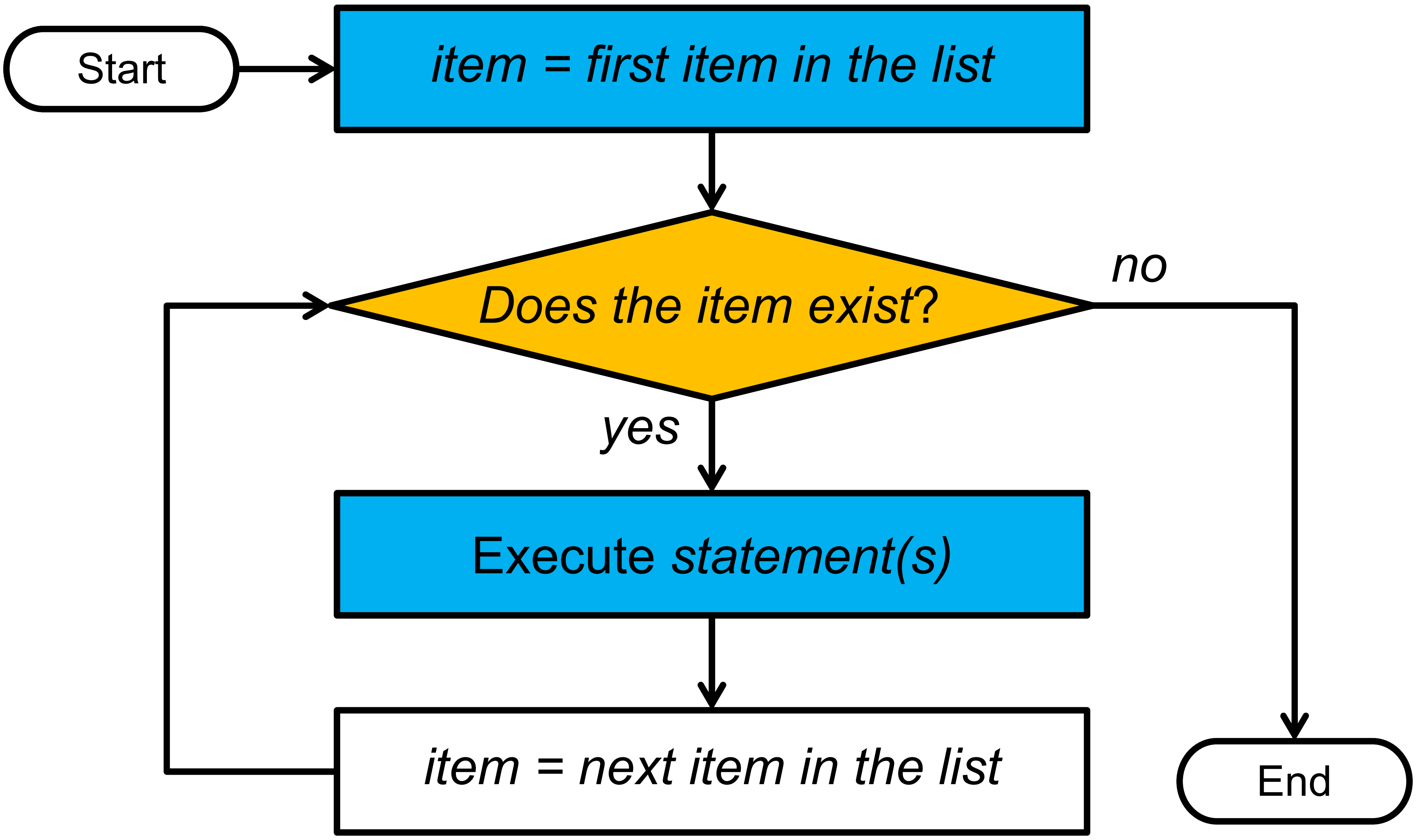
For Loops

`for` *item* `in` *a list of items* `:`

. . .*statement(s)*. . .

- The *statement(s)* are executed for each item
- For example, if there are 10 items, the *statement(s)* will be executed 10 times

The Flow of a For Loop


Start
item = first item in the list
Does the item exist?
no
yes
Execute *statement(s)*
item = next item in the list
End

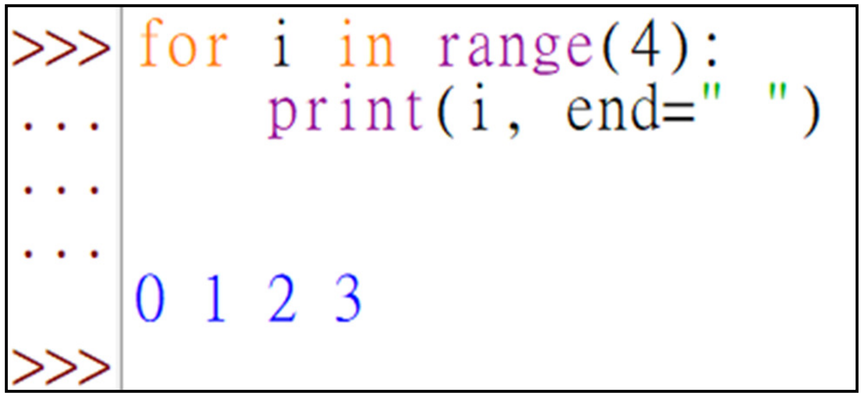
Using Range in a For Loop

- You need to give a list of items to a for loop
- This is where the range command is commonly used
- For example, the following code prints out a list of four numbers:

```
for i in range(4) :
     print(i, end=" " )
```

This creates four items: 0, 1, 2 and 3

end=" " means a space is put at the end of the number when it is printed, instead of going to the next line

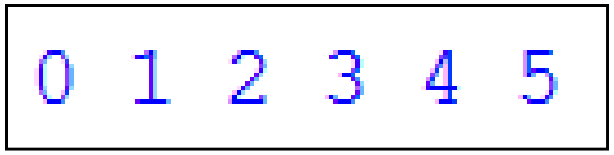


```
>>> for i in range(4):
        print(i, end=" ")
...
...
...
0 1 2 3
>>>
```

Controlling a For Loop

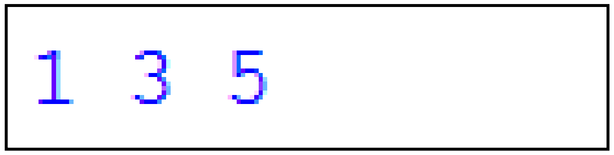
- As you can see from the previous slide the range command can be used to control:
  - how many times the content of a for loop is repeatedly executed
  - the number each time the loop content is executed
- Here are some more examples:

```
for i in range(0, 6):
    print(i, end=" ")
```



```
0 1 2 3 4 5
```

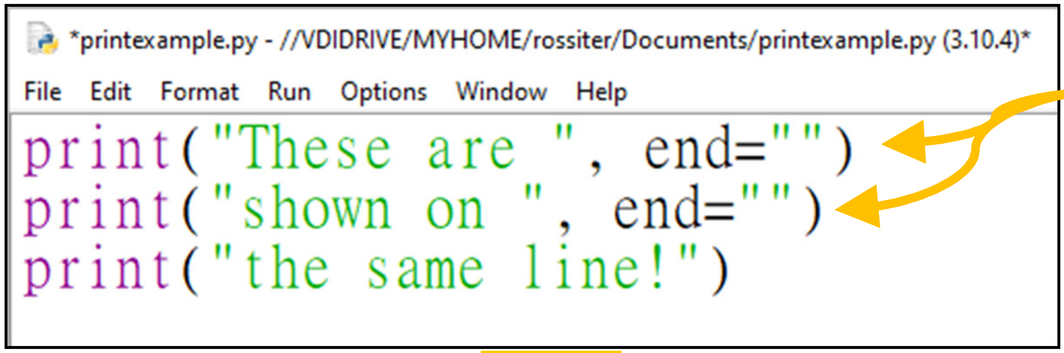
```
for i in range(1, 6, 2):
    print(i, end=" ")
```



```
1 3 5
```

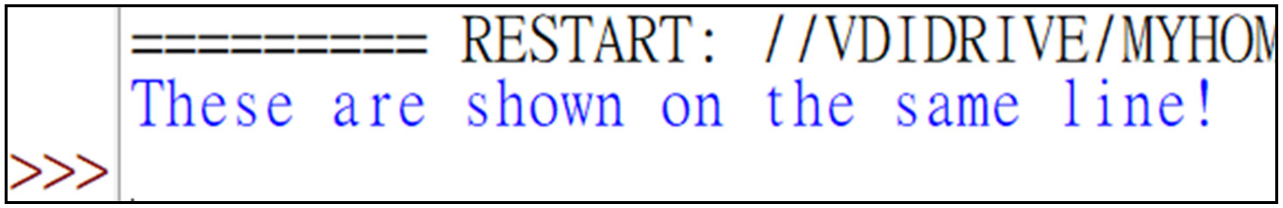
Printing Things Using ‘end=’

- In the previous examples, we used ‘end =’ to ask the print command to print a space when it has finished
- This is useful when you have multiple print commands and you want them to print on the same line, e.g:


printexample.py - //VDIDRIVE/MYHOME/rossiter/Documents/printexample.py (3.10.4)
File Edit Format Run Options Window Help

```
print("These are ", end="")
print("shown on ", end="")
print("the same line!")
```

Here nothing, i.e. "", is printed at the end (there’s no space in "")



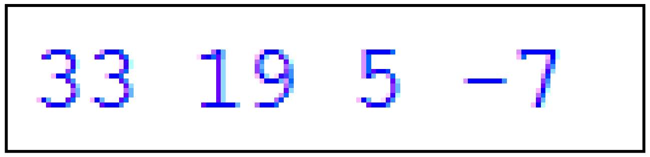
```
========= RESTART: //VDIDRIVE/MYHOM
These are shown on the same line!
>>>
```

- *Here we are using a program instead of the shell*

Using a ‘Fixed’ List in a For Loop

- You can use a ‘fixed’ list
- For example, you can use any numbers you like:

```
for i in [33, 19, 5, -7]:
    print(i, end=" ")
```

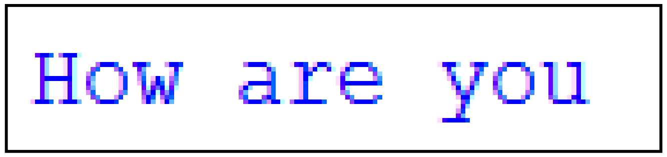


```
33 19 5 -7
```

You use a pair of brackets, i.e. `[]`*, to enclose a list of items*

- Or you can choose not to use any numbers at all:

```
for word in ["How", "are", "you"]:
    print(word, end=" ")
```



```
How are you
```