Python Turtle - Lesson 2

Topics

In this lesson you will:

- learn how to use iteration to reduce your code length
- learn how to represent programs in a flowchart
- write turtle programs using a `for` loop

Part 1: Iteration introduction

Sequential flow

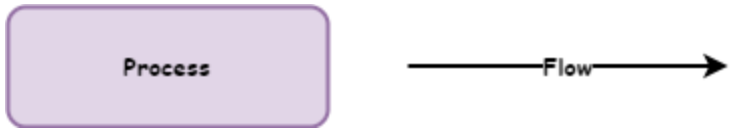
Python has been executing each line of our code one after another

- called *sequential*
- the default way that programs work
- the movement of a program called *the flow* of the program (like water, or electricity)

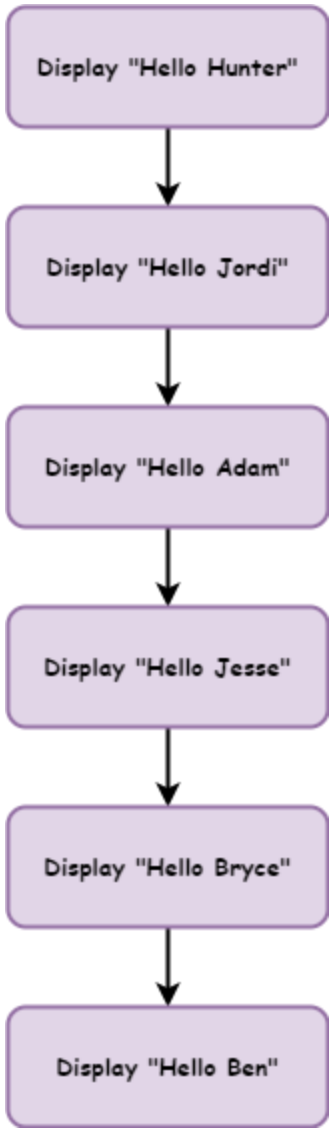
Introduction to flowcharts

What are flowcharts?

- special diagram used to show the flow of a computer program
- show each process in a program and how the program moves from one process to the next
- Symbols
    - rectangles represent processes
    - arrows represent the flow.



A program to say hello to six people would be represented like this:



Transferring this code to Python would produce the following code:

```
# our itreation program

print("Hello Hunter")
print("Hello Jordi")
print("Hello Adam")
print("Hello Jesse")
print("Hello Bryce")
print("Hello Ben")
```

Sequential flow: line `1` → line `8`.

Changing the order of the code will produce different results.

```
# our itreation program

print("Hello Jesse")
print("Hello Bryce")
print("Hello Ben")
print("Hello Hunter")
print("Hello Jordi")
print("Hello Adam")
```

Sequential code limitations

Sequential programming starts to become a problem with larger programs.

- saying hello to 500 or 1,000 people
- changing the code from 'hello' to 'good morning'

Sequential coding is not *scalable*

Iteration

There is a lot of repetition in the code

```
# our itreation program

print("Hello Jesse")
print("Hello Bryce")
print("Hello Ben")
print("Hello Hunter")
print("Hello Jordi")
print("Hello Adam")
```

Lines `3` to `8` → the same line with small changes

Clashes with the DRY programming principle.

DON'T

REPEAT

YOURSELF

To not repeat yourself, use *iteration* (often called *loops*).

- repeat the same code with a slight change each time
- repeat the code `print("Hello", name)` with a different name each time

For loops

Our first *control structure*

- control the flow of the program
- cause it to deviate from its default sequential flow

Change your code to the code below:

```
# our itreation program

names = ["Hunter", "Jordi", "Adam", "Jesse", "Bryce", "Ben"]

for name in names:
    print("Hello", name)
```

PRIMM

- *predict* what you think will happen
- *run* the code.

Investigate the code:

```
names = ["Hunter", "Jordi", "Adam", "Jesse", "Bryce", "Ben"]
```

- is a *list* which works just like a real world list
- the `[` and `]` indicate the beginning and end of the list.
- `"Hunter"`, `"Jordi"`, `"Adam"`, `"Jesse"`, `"Bryce"`, `"Ben"` items in the list (*elements*)
- commas `,` separate elements
- use `names =` to call the list `names`

```
for name in names:
```

- how we create `for` loops
- `for` keyword identifies the beginning of a `for` loop
- `in names` tells Python to repeat the code below using each *element* of the `names` list
- `name` refers to the current `names` element being used.

`print("Hello", name)`

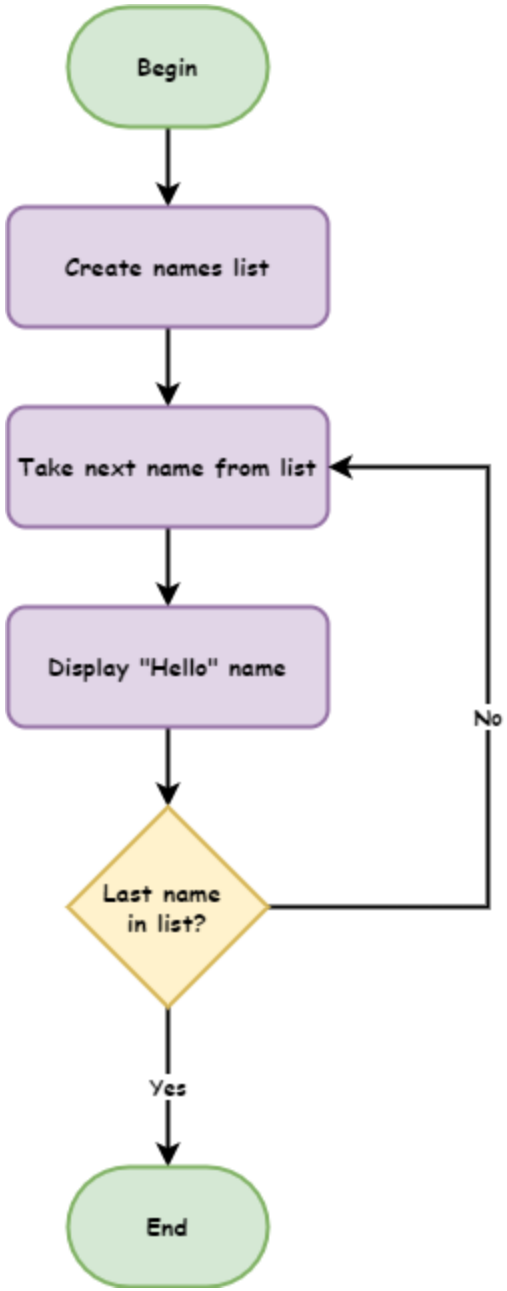
- indentation below the `for` loop
  - the code that needs to be repeated
  - can be multiple lines (*block*)
  - indents should be four spaces
    - use the `tab` key
- `print("Hello", name)`
  - print `Hello` to the **Shell**
  - `name` : current element taken from the `names` list

Flowchart

Two new flowchart symbols:

- *Terminators*: these represent the the beginning and end of your code
- *Decisions*: these are questions the program need to answer, and will result in the flow splitting into multiple branches.

`for` loop flowchart



Tracing with debugger

Use Thonny's debugger to follow `for` loops

Launch by clicking the bug button

Press **F7** to go through the code step by step.

Note of the values in the **Variables** panel.

Code blocks

How do code blocks work

Change your code so it is the same as below:

```
# our itreation program

names = ["Hunter", "Jordi", "Adam", "Bryce", "Ben"]

for name in names:
    print("Hello", name)
    print("How are you?")
```

Predict what you think the code will do and then *run* it.

Notice all the code block is repeated.

- the `for` loop repeats all the lines of code at the same level of indentation
- ensure that the code block uses the same number of spaces

What happens if we remove the indentation?

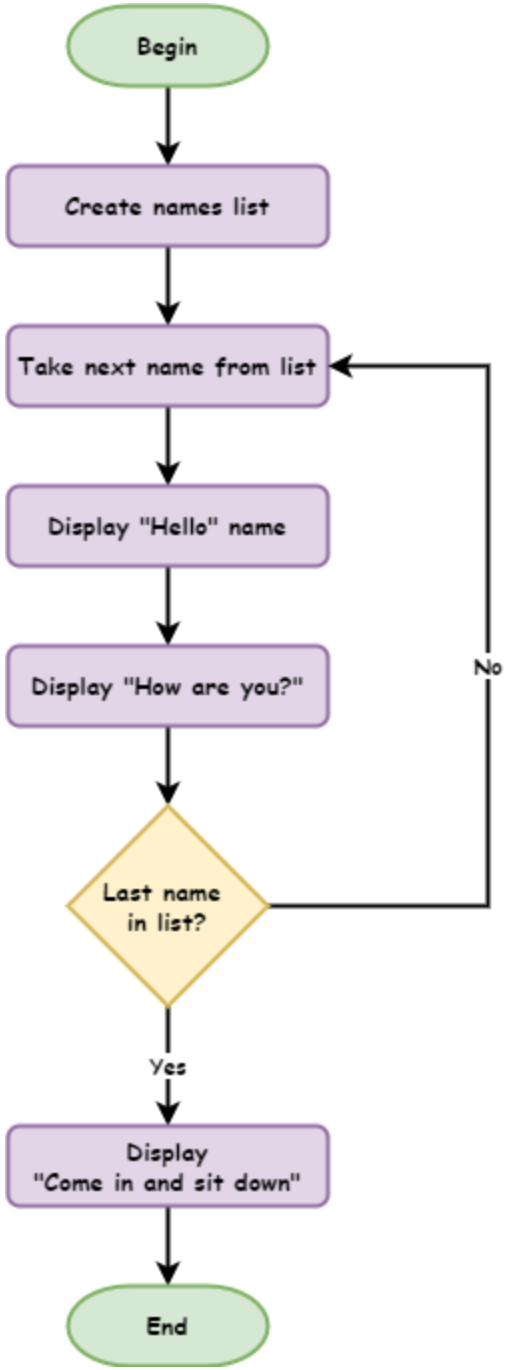
Change your code so it looks like the code below:

```
# our itreation program

names = ["Hunter", "Jordi", "Adam", "Bryce", "Ben"]

for name in names:
    print("Hello", name)
    print("How are you?")

print("Come in and sit down")
```

Predict and *run* your code.

`print("Come in and sit down")` is not repeated

- it is not indented so it is not part of the `for` loop
- it runs after the `for` loop is finished


Begin
Create names list
Take next name from list
Display "Hello" name
Display "How are you?"
Last name in list?
No
Yes
Display "Come in and sit down"
End

Part 2: List numbers and Range

You can also run loops over lists of numbers.

Try the code below:

```
number_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for number in number_list:
    print(number)
```

Change the code to print the numbers between 1 and 100

`range` function → make list between two given numbers

Try the code below:

```
number_list = range(1,101)

for number in number_list:
    print(number)
```

Unpack the code:

- `range` → create a list of numbers
- `1` → first number in the list
- `101` → first number *not* in the list

Can use the `range` function directly in `for` loop

```
for number in range(1,101):
    print(number)
```

Use for Turtle

Code blocks can be made up of any code, including Turtle code.

Create a new file and type in the code below.

```
import turtle

window = turtle.Screen()
window.setup(500,500)

my_ttl = turtle.Turtle()

for number in range(1,101):
    my_ttl.forward(100)
    my_ttl.backward(100)
    my_ttl.left(3)
```

PRIMM:

- *Predict* what you think will happen
- *Run* the code. Did it do what you predicted?
- *Investigate* the code by changing aspects of the code.
- *Modify* the code so that is makes a complete circle.

```
##################################
## Draw a Triangle in 3 lines ##
##################################
```

After line `9`, as the comments says, write code that will create a triangle but only use 3 lines to do this.

Exercise 3

Create a new file and save it in your subject folder calling it **`lesson_2_ex_3.py`**. Then type the following code into it.

```
import turtle

window = turtle.Screen()
window.setup(500, 500)
my_ttl = turtle.Turtle()

##################################
## Draw a Hexagon in 3 lines ##
##################################
```