COSC2429 Introduction to Programming

Python Turtle Graphics and Python Modules

Outline

- Hello, little turtles!
- Our first turtle program
- More turtle program
- Instances
- The **for** loop
- Flow of execution of the for loop
- Iteration simplifies our turtle program
- The **range** function
- A few more turtle methods and observations
- Modules
- The **math** module
- The **random** module

Hello, little turtles!

- There are many **modules** in Python that provide very powerful features that we can use in our own programs.
  - Some can send email or fetch web pages.
  - Others allow us to perform complex math calculations.
- A **module** is a Python file containing definitions and statements.
- We now introduce a module that allows us to create a data object called a **turtle** that can be used to draw pictures.
  - A turtle can follow simple commands such as go **forward** and turn **right**.
  - As the turtle moves around, if its tail is **down** touching the ground, it will draw a line (leave a trail behind) as it moves.
  - If you tell your turtle to lift **up** its tail it can still move around but will not leave a trail.
  - You will be able to make some pretty amazing drawings with these simple capabilities.

Our first turtle program

- Let's try to create a new turtle and start drawing a simple figure like a rectangle.

```
import turtle                 # allows us to use the turtles module

win = turtle.Screen()         # creates a graphics window
tito = turtle.Turtle()        # creates a turtle named tito

tito.forward(150)             # ask tito to move forward 150 pixels
tito.left(90)                 # ask tito to turn left 90 degrees
tito.forward(75)              # complete the second side of a rectangle
…
```

- Does the program run too fast? No problem, add the following line after creating the graphics window:

```
win.delay(100)                # delay each action that tito performs by 100 ms
```

More turtle program

- An object can have various **methods** — things it can do — and it can also have **attributes** (sometimes also called **properties**). For example:
  - Each turtle has a *color* attribute that can be set to a specific color.
  - It is similar with the turtle *pensize* or the window's *bgcolor*.

```
import turtle

# Setup a window and a turtle object to draw
win = turtle.Screen()
win.bgcolor("lightgreen")
tess = turtle.Turtle()
tess.color("blue")
tess.pensize(3)

# Draw an angle
tess.forward(80)
tess.left(120)
ttess.forward(80)

win.exitonclick()                    # wait for a user click on the canvas
```

Instances

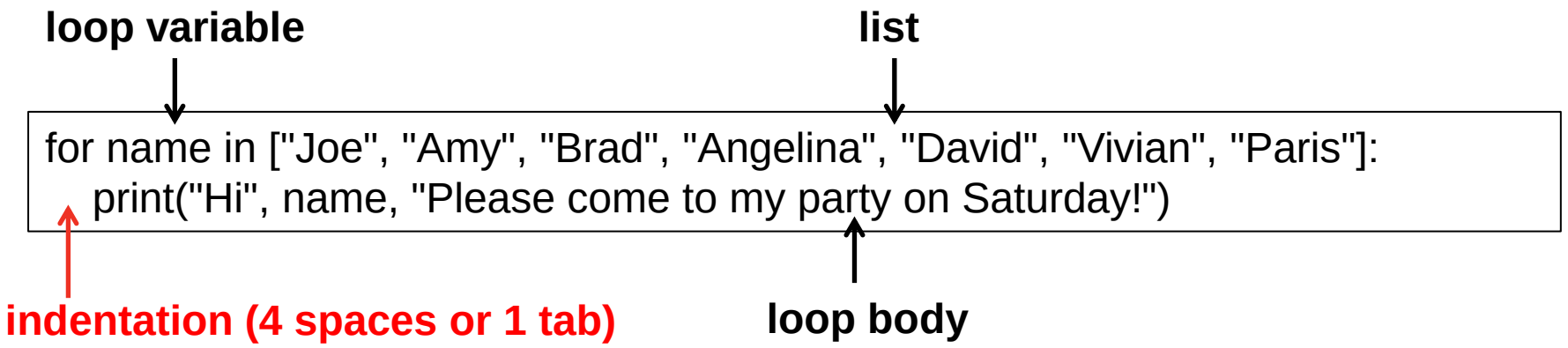
- Just like we can have many different integers in a program, we can have many turtles. Each of them is an independent object and we call each one an **instance** of the Turtle type (class).
- Each instance has its own attributes and methods — so tito might draw with a thin black pen and be at some position, while tito might be going in her own direction with a fat pink pen.
- The following program is what happens when tito draws a square and tess draws a triangle:

Instances

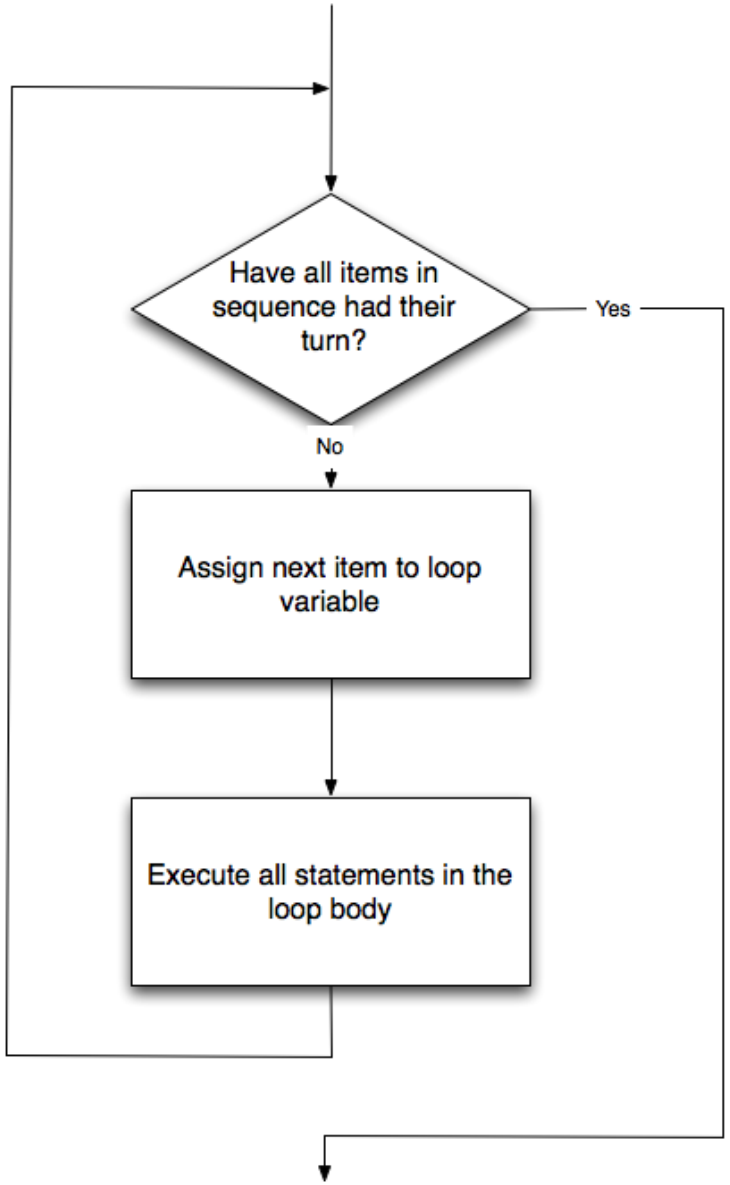
```
import turtle

# Setup a window and two turtles to draw
win = turtle.Screen()
win.bgcolor("lightgreen")
tito = turtle.Turtle()
tess = turtle.Turtle()
tess.color("hotpink")
tess.pensize(5)

# tito draws a square
tito.forward(150)
tito.left(90)
tito.forward(150)
tito.left(90)
tito.forward(150)
tito.left(90)
tito.forward(150)
tito.left(90)

# tess draws an equilateral triangle
tess.forward(80)
tess.left(120)
tess.forward(80)
tess.left(120)
tess.forward(80)
tess.left(120)

win.exitonclick()
```

The **for** loop

- When we drew the square, it was quite tedious. We had to move then turn, move then turn, etc. four times. If we were drawing a polygon with 42 sides, it would have been a nightmare to duplicate all that code.
- It would be nice to be able to repeat some code over and over again. We refer to this repetitive idea as **iteration**.
- In Python, the **for** statement allows us to write iteration.
- As a simple example, let's say we have some friends, and we'd like to send each of them an email inviting them to our party. At the moment, we don't quite know how to send email yet, so we'll just print a message for each friend.



Flow of execution of the **for** loop




```
for loop_variable in list:
    loop_body_stmts
```

Iteration simplifies our turtle program

```
import turtle

# Set up a window and a turtle
win = turtle.Screen()
win.bgcolor("lightgreen")
tito = turtle.Turtle()
tito.pensize(5)

# Draw a side 4 times to make a square
for i in [0, 1, 2, 3]:              # i is a common abbreviation for integer
    tito.forward(150)
    tito.left(90)

win.exitonclick()
```

Iteration simplifies our turtle program

- Actually, it's more important that we found a **repeated pattern** rather than saving a few lines of code
- We could use another kind of list to do the same thing

```
import turtle

# Set up a window and a turtle
win = turtle.Screen()
win.bgcolor("lightgreen")
tito = turtle.Turtle()
tito.pensize(5)

# Draw a side 4 times to make a square
for a_color in ["yellow", "red", "purple", "blue"]:
    tito.forward(150)
    tito.left(90)

win.exitonclick()
```

Iteration simplifies our turtle program

- Now, let's add one more line of code to make the square more beautiful ☺

```
import turtle

# Set up a window and a turtle
win = turtle.Screen()
win.bgcolor("lightgreen")
tito = turtle.Turtle()
tito.pensize(5)

# Draw a side 4 times to make a square
for a_color in ["yellow", "red", "purple", "blue"]:
    tito.color(a_color)
    tito.forward(150)
    tito.left(90)

win.exitonclick()
```

The **range** function

- Generating a list of integers is a very common thing to do, especially when using **for** loop
- The **range** function helps to generate a list of integers very effectively
- For example, the following code will print 0, 1, 2, 3

```
for i in range(4):              # range(stop)
    print(i)
```

- How's about this?

```
for i in range(1, 5):           # range(start, stop)
    print(i)
```

- And this?

```
for i in range(1, 20, 2):       # range(start, stop, step)
    print(i)
```

- Quick check: what is the command to generate [2, 5, 8]?

A few more turtle methods and observations

- A turtle's tail can be picked up or put down. This allows us to move a turtle to a different place without drawing a line.

```
tito.up()              # ask tito to pick her tail up, i.e. no draw after that
tito.forward(100)      # this moves tito, but no line is drawn
tito.down()            # ask tito to put her tail down
```

- Every turtle can have its own shape. The ones available "out of the box" are arrow, blank, circle, classic, square, triangle, turtle.

```
...
tito.shape("turtle")
...
```

- You can speed up or slow down the turtle's animation speed. Speed settings can be set between 1 (slowest) to 10 (fastest).

```
tito.speed(10)
```

A few more turtle methods and observations

- A turtle can “stamp” its footprint onto the canvas, and this will remain after the turtle has moved somewhere else. Stamping works even when the pen is up.
- Let’s do an example that shows off some of these new features.

```
import turtle

win = turtle.Screen()
win.bgcolor("lightgreen")
tess = turtle.Turtle()
tess.color("blue")
tess.shape("turtle")

tess.up()
for size in range(5, 60, 2):          # start with size = 5 and grow by 2
    tess.stamp()                      # leave an impression on the canvas
    tess.forward(size)                # move tess along
    tess.right(24)                    # and turn her

win.exitonclick()
```

*** Summary of turtle methods ***

Method	Parameters	Description
Turtle	None	Creates and returns a new turtle object
forward	distance	Moves the turtle forward
backward	distance	Moves the turle backward
right	angle	Turns the turtle clockwise
left	angle	Turns the turtle counter clockwise
up	None	Picks up the turtles tail
down	None	Puts down the turtles tail
color	color name	Changes the color of the turtle's tail
fillcolor	color name	Changes the color of the turtle will use to fill a polygon
heading	None	Returns the current heading
position	None	Returns the current position
goto	x, y	Move the turtle to position x, y
begin_fill	None	Remember the starting point for a filled polygon
end_fill	None	Close the polygon and fill with the current fill color
dot	None	Leave a dot at the current position
stamp	None	Leaves an impression of a turtle shape at the current location
shape	shapename	Should be 'arrow', 'classic', 'turtle', or 'circle'

Modules

- A **module** is a file containing Python definitions and statements intended for use in other Python programs.
- There are many Python modules that come with Python as part of the **standard library** and you can write your own modules too if you want.
- We have already used one of these modules extensively - the **turtle** module.
- Once we **import** a module, we can use things that are defined inside that module.

```
import turtle
```

- The Python Documentation site for Python 3.5 is an extremely useful reference for all aspects of Python.
- You should browse through the sections in this site to get familiar with the information that it provides.
- The “Quick search” function in this site is very handy.

The **math** module

- The math module contains the math functions you would typically find on your calculator and some math constants like pi and e

```
import math

print(math.pi)
print(math.e)
print(math.sqrt(2.0))
print(math.sin(math.radians(90)))   # sin of 90 degrees
```

- Notice another difference between this module and our use of **turtle**.
  In **turtle** we create objects (either Turtle or Screen) and call methods on those objects.
- Math functions do not need to be constructed. They simply perform a task. They are all housed together in a module called math.

The **random** module

- We often want to use **random numbers** in programs. Here are a few typical uses:
  - To play a game of chance where the computer needs to throw some dice, pick a number, or flip a coin
  - To shuffle a deck of playing cards randomly
  - To randomly allow a new enemy spaceship to appear and shoot at you
- Python provides a module random that helps with tasks like this. You can take a look at it in the documentation.
- Here is an example of the key things we can do with the random module.

```
import random

# get a random floating number in [0.0, 1.0)
prob = random.random()
print(prob)

# return a random integer in [1, 7)
dice_value = random.randrange(1, 7)
print(dice_value)
```

Exercise

Write a program that generates and prints out a random value of a dice. The program should do this five times and then exits.