INFT1004 - SEMESTER 1 - 2017		LECTURE TOPICS	
Week 1	Feb 27	Introduction, Assignment, Arithmetic	
Week 2	Mar 6	Sequence, Quick Start, Programming Style	
Week 3	Mar 13	Pictures, Functions, Media Paths	
Week 4	Mar 20	Arrays, Pixels, For Loop, Reference Passing	
Week 5	Mar 27	Nested Loops, Selection, Advanced Pictures	
Week 6	Apr 3	Lists, Strings, Input & Output, Files	Practical Test
Week 7	Apr 10	Drawing Pictures, Program Design, While Loop	Assignment set
Recess	Apr 14 – Apr 23	Mid Semester Recess Break	
Week 8	Apr 24	No Lecture / Revision and Assignment in Labs	
Week 9	May 1	Data Structures, Processing sound	
Week 10	May 8	Advanced sound	Assignment part 1 due 8:00am Tue, May 9
Week 11	May 15	Movies, Scope, Import	
Week 12	May 22	Turtles, Writing Classes	Assignment part 2 due 8:00am Tue, May 23
Week 13	May 29	Revision	
Mid Year Examination Period - MUST be available normal & supplementary period			

Lecture Topics and Lab topics are the same for each week

INFT1004 - SEMESTER 1 - 2017		LECTURE TOPICS	
Week 1	Feb 27	Introduction, Assignment, Arithmetic	
Week 2	Mar 6	Sequence, Quick Start, Programming Style	
Week 3	Mar 13	Pictures, Functions, Media Paths	
Week 4	Mar 2		
Week 5	Mar 2		
Week 6	Apr 3		
Week 7	Apr 1		t
Recess	Apr 14 -		
Week 8	Apr 2		
Week 9	May		
Week 10	May		due 9
Week 11	May		
Week 12	May 2		due 23
Week 13	May 29	Revision	
Mid Year Examination Period - MUST be available normal & supplementary period			

Lecture Topics and Lab topics are the same for each week

Assignment is still due on Tuesday 23 May at 8:00 am (today)

No late penalty if received before Friday 26 May – 8:00am

But will be 4 days late if received after this time.

INFT1004

Visual Programming

Module 12.1
Turtles and other classes

Guzdial & Ericson - Third Edition – chapter 16
Guzdial & Ericson - Fourth (Global) Edition – chapter 17

Object-oriented programming

Right from the beginning we've been dealing with objects:

- Pictures
- Pixels
- Colours
- Sounds
- Samples . . .

However, our principal unit of programming has been the function

Object-oriented programming

In object-oriented programming the principal unit of programming is the class

A class defines what an object will look like: what properties (data) it has and what methods it has

methods(functions, procedures) – what it can do

Picture Class

eg. picture class

has properties such as
height, width, pixels, etc

and methods such as
repaint and *explore, etc*

Picture
height width pixels[] . . .
repaint() explore() . . .

Sound Class

eg. sound class

has properties such as
duration, samples, etc

and methods such as
play and *explore, etc*

Sound
duration samples[] . . .
play() explore() . . .

Object-oriented programming

Its time we saw how to define and use our own classes

Mod12_01_SmartTurtleClass.py

But first, turtles!

A turtle is a little animated picture that can move around in a world, drawing lines with a pen

Turtles have been used for many years to teach children the elements of programming

Turtles have been included in JES

But first, turtles!

The canvas on which a turtle moves and draws is called a world

```
earth = makeWorld()
```

Once we have a world, we can put turtles in it

```
liner = makeTurtle(earth)
```

(Try these in the command window of JES)

Proper OO notation

`makeWorld()` and `makeTurtle()` are JES functions, like `makePicture()` and `makeSound()`

Correct object-oriented syntax for the same thing is

```
earth = World()
liner = Turtle(earth)
```


Proper OO notation

`makeWorld()` and `makeTurtle()` are JES functions, like `makePicture()` and `makeSound()`

Correct object-oriented syntax for the same thing is

```
earth = World()
liner = Turtle(earth)
```

These instructions make `earth` a new `World`, and make `liner` a new `Turtle` on the `World` called `earth`

We'll now stick with this Object Oriented notation

What can turtles do?

Here are some of the things turtles can do.

```
liner.forward()
liner.forward(pixels)
liner.turnLeft()
liner.turnRight()
liner.turn(degrees)
liner.penUp()
liner.penDown()
liner.setColor(red)
liner.setPenWidth(pixels)
liner.drop(picture)
```

Mod12_01_TestTurtles.py

What can turtles do?

Here are some of the things turtles can do.

```
liner.forward()
liner.forward(pixels)
liner.turnLeft()
liner.turnRight()
liner.turn(degrees)
liner.penUp()
liner.penDown()
liner.setColor(red)
liner.setPenWidth(pixels)
liner.drop(picture)
```

To learn, or remember what they achieve, either look in the book or experiment with them (have fun)

Mod12_01_TestTurtles.py

Making our own classes

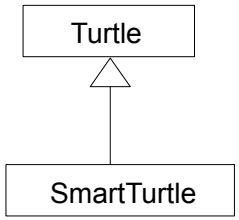
What if we want to add features to turtles?

Making our own classes

What if we want to add features to turtles?

We can't easily modify the Turtle class; but we can make a new class based on the Turtle class;

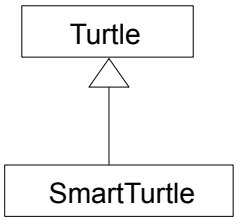
This is called a 'subclass' of Turtle



Mod12_01_SmartTurtleClass.py

Making our own classes

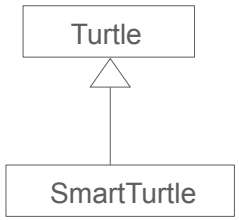
A subclass has all the features of its 'parent' class, but we can easily add more features



Mod12_01_SmartTurtleClass.py

Making our own classes

A subclass has all the features of its 'parent' class, but we can easily add more features



See the SmartTurtle class in Mod12_01_SmartTurtleClass.py

As always in Python, the indentation defines the start and end of the class

Classes have methods

A 'function' defined within class is called a method

When we create an object of this class, we can call the method by

```
objectName.methodName(parameters)
```

```
liner.turn(degrees)
```

Note we have seen this notation with the string class already e.g. `aString.lower()`

Classes have methods

```
def methodName(self, parameter1, parameter2, ..)
```

Note that every method of a class must have the parameter **`self`**, which refers to the object for which the method has been called

Classes have methods

```
def methodName(self, parameter1, parameter2, ..)
```

Note that every method of a class must have the parameter **self**, which refers to the object for which the method has been called

When the method is called, there is no argument corresponding to this first parameter called **self**

```
methodName(argument1, argument2, ..)
```


Classes have methods

```
def methodName(self, parameter1, parameter2, ..)
```

A method can be given additional parameters as required

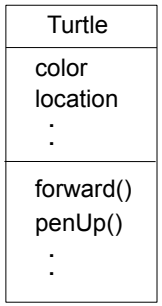
These parameters will have corresponding arguments that are used when calling the method.

Classes have data

Classes can have data, which look just like variables

Each object of the class gets its own copy of the data

For example, each turtle has its own color and location



Using the methods and data

Remember, a class is the code that defines the data and the methods that each object of the class will have

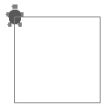
Within the code for the class, if we want an object to execute one of its own methods we write `self.methodName(arguments)`

and if we want an object to refer to its own data we write `self.variableName`

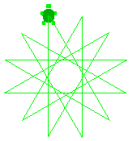
Using the methods and data

```
myWorld = World()
leonardo= SmartTurtle(myWorld)

leonardo.drawSquare()
```



```
leonardo.drawStar(12)
```



Mod12_01_TestTurtles.py
Mod12_01_SmartTurtleClass.py

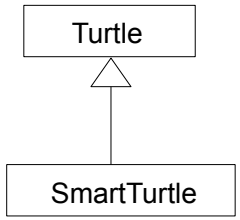
Using the methods and data

```
file = pickAFile()
barbara = makePicture(file)
leonardo.pictureSwirl(barbara)
```

Mod12_01_TestTurtles.py
Mod12_01_SmartTurtleClass.py

A completely new class

SmartTurtle is a subclass of the Turtle class



There will be times when we want to create a completely new class (not a subclass) see Mod - Writing a New Class

INFT1004

Visual Programming

Module 12.2
Writing your own Class

Guzdial & Ericson - Third Edition – chapter 16
Guzdial & Ericson - Fourth (Global) Edition – chapter 17

A completely new class

Slide is an example of a totally new class
(not a subclass of an existing class)

Slide

Mod12_2_SlideClass.py

A completely new class

Slide is an example of a totally new class
(not a subclass of an existing class)

Slide

Note that class names conventionally start with a capital letter

Mod12_02_SlideClass.py

A completely new class

A Slide object will have two data items, a picture and a sound

It will have a method, called `show()`, which will show the picture and a method and plays the sound

Slide
pictureFile soundFile
setPicture() getPicture() setSound() getMySound() show()

Constructor – a special method

When we create a new object of a class

```
slide = Slide()
```

Python calls a special method of the class called a constructor

Constructor – a special method

```
slide = Slide()
```

A constructor

- creates the new object
- it sets aside space for it
- associates the name with that space
- associates the name with the specified class type
- and possibly initialises the data

Constructor – a special method

```
slide = Slide()
```

In Python, the constructor for a class actually has the name:

`__init__()`

but this name is never called

Constructor – optional arguments

In a constructor (and in fact in any method), you can provide 'default' values for the parameters

by writing `parameterName=value` rather than just `parameterName`

```
def __init__(self, pictureFile=None, soundFile=None):
```

Mod12_02_SlideClass.py

Constructor – optional arguments

In a constructor (and in fact in any method), you can provide 'default' values for the parameters

by writing `parameterName=value` rather than just `parameterName`

```
def __init__(self, pictureFile=None, soundFile=None):
```

If the function is called without the argument, the parameter is given the default `value`

Mod12_02_SlideClass.py

Constructor – optional arguments

Some Python programmers use the predefined value of `None` as a standard default value for parameters

Example

Mod12_02_SlideClass.py

In the `Slide` class, if the constructor is called without a picture file or a sound file, it makes a slide with the church picture and the church sound by default

(assuming you have setMediaPath correctly)

Constructor

```
def __init__(self, pictureFile = None, soundFile = None):

   if pictureFile == None:
      self.setPicture(makePicture(getMediaPath("church.jpg")))
   else:
      self.setPicture(makePicture(picFile))

   if soundFile == None:
      self.setSound(makeSound(getMediaPath("church.wav")))
   else:
         self.setSound(makeSound(soundFile))
```

In the `Slide` class, if the constructor is called without a picture file or a sound file, it makes a slide with the church picture and the church sound by default

Accessors and mutators

Ideally, an object's data is accessed only through its own methods

An accessor / getter is a method that gives us the value of a data item; it's generally called get()

A mutator /setter is a method that changes the value of a data item; it's generally called set()

Accessors and mutators

In some programming languages the program can insist that data be accessed by get() and set()

We can't enforce this in Python; instead we provide them, and hope that programmers will respect this design ideal

Remember this is best practice – you should always use get() and set()

get and set

```
# Set and get the Slide's picture property
  def setPicture(self, newPicture):
     self.picture = newPicture

  def getPicture(self):
     return self.picture


  # Set and get the Slide's sound property
  def setSound(self, newSound):
     self.sound = newSound

  def getMySound(self):
     return self.sound
```


Polymorphism

Polymorphism, one of the buzz words of OO programming, has a number of different but related meanings

Literally, the word means 'taking many forms'

The method `show()` is an example of one form of polymorphism, also called method overloading

Polymorphism

The same name, `show()`, does different things depending on its arguments; that is, there are different forms of the method

Eg. for pictures, sounds and slides

Just to emphasise this, the `show()` for the `Slide` class uses the `show()` for the picture class

How is `explore()` polymorphic?

show()

```
def show(self):
   show(self.getPicture())
   blockingPlay(self.getMySound())
```

Mod12_02_SlideClass.py

Using the Slide Class

```
def playSlideShow():

   slide = {}  # This makes slide an empty list

   slide[0] = Slide(getMediaPath("barbara.jpg"),
               getMediaPath("bassoon-c4.wav"))
   slide[1] = Slide(getMediaPath("beach.jpg"),
               getMediaPath("bassoon-e4.wav"))
   slide[2] = Slide(getMediaPath("church.jpg"),
               getMediaPath("bassoon-g4.wav"))
   slide[3] = Slide(getMediaPath("jungle2.jpg"),
               getMediaPath("bassoon-c4.wav"))

   # Now we display each Slide in the list
   for i in range(0,4):
      slide[i].show()
```

Mod12_02_SlideClass.py