CSCI120

Introduction to Computer Science I using Python 3

Introduction to Object-Oriented Programming

Introduction to Object-Oriented Programming (OOP)

Python is a *structured programming language* using *sequences* (statements executed in sequence), *decisions* (if) and *looping* (for and while) organized to aid program understanding and modification.

Introduction to Object-Oriented Programming (OOP)

Python supports multiple programming styles (also known as *paradigms*), including *imperative* (issuing command statements to tell the computer what to do) and *procedural* (using programmer-defined functions to organize and simplify programs).

Introduction to Object-Oriented Programming (OOP)

Procedural programming includes the imperative and structured programming paradigms.

Introduction to Object-Oriented Programming (OOP)

In this Python course we started with structured imperative programming and went on to procedural programming when we used programmer-defined functions...

Introduction to Object-Oriented Programming (OOP)

We are now about to learn how to use a very important programming method - OOP

Introduction to Object-Oriented Programming (OOP)

The Object-Oriented Programming (OOP) paradigm was first introduced at MIT in the 1950s.

Introduction to Object-Oriented Programming (OOP)

The idea of object-oriented programming gained momentum in the 1970s, and in the early 1980s Bjarne Stroustrup integrated object-oriented programming into the C language. The resulting language was called C++ and it became the first object-oriented language to be widely used commercially.

Introduction to Object-Oriented Programming (OOP)

In the early 1990s a group at Sun Microsystems led by James Gosling developed a simpler version of C++ called *Java* that was developed into a language for programming Internet applications. The language gained widespread popularity as the Internet boomed and was the language chosen by Google to develop mobile Android apps.

Object-Oriented Programming

The Python language was first developed in 1989 by Guido van Rossum and by 1990 had the capability for object-oriented programming...

What is OOP?
What is an Object?

What is Object-Oriented Programming?

To know what OOP is, you must first know what an object is. . .

An object . . .

an object is an entity (a thing that exists)

Examples of objects

We have already been using many different types of object on this course, for example we have used objects of type:

int	float
str	bool
list	file

How to make an int object

```
age = 10
print(type(age))
```

```
<class 'int'>
```

How to make a float object

```
temperature = 25.6
print(type(temperature))
```

```
<class 'float'>
```

How to make a str (string) object

```
myname = 'Anne Dawson'
print(type(myname))

<class 'str'>
```

How to make a bool object

```
passed = True
print(type(passed))
```

```
<class 'bool'>
```

How to make a list object

```
scores = [91,88,78,94,57,69]
print(type(scores))
```

```
<class 'list'>
```

A data type is a class!

Notice that all the built-in data types (int, float, str, list etc) are known as classes...

A class has actions associated with it – for example, with an int or float object you can use + , -, * and /

A data type is a class!

The actions associated with objects of class `list`, `str` and `file` have actions built in to their *methods*...

```
mylist.sort()
mystring.capitalize()
myfile.close()
```

A class's methods can change the state of the object

For example, the data in a list are sorted by the `sort()` method, the letters in a string are converted to uppercase using the `upper()` method.

```
mylist.sort()
mystring.upper()
```

In summary, class objects have...

their own name, (e.g. `firstname`), their own data, (e.g. `'Anne'`) and their own methods, (e.g. `upper()`)

and every object belongs to a class...

An object is said to be an *instance* of a class

Python lets you create your own classes...

Just like there are built-in functions like `print()`, `str()` and functions you make yourself (programmer-defined functions), their are built-in classes and *classes you can define yourself...*

Why would you want to create your own classes?

As your programs get bigger, keeping track of data gets more and more difficult...

Say you're storing data on students on a Computer Science course, you could use a list to store all the pertinent data for name, course code, student number, scores etc...

Why would you want to create your own classes?

After using the program for a while you may decide to store extra student data in your list for date of birth for example...

You may need to amend your program on several lines to make allowance for this change...

Why would you want to create your own classes?

If you're working with a team of programmers and you hand your program on to another programmer for maintenance, it would be difficult for them to try to figure out what data you're storing and where it is stored and manipulated.

Creating your own Student class simplifies your program...

Creating your own Student class simplifies your program...

A Student class can be used to make Student objects, each with its own set of data encapsulated in the object itself, and with the ability to call any of the methods specially written for that class.

Objects

Computer-programmed objects are similar in many respects to everyday objects . . .

such as cars, computers, cell phones and music systems . . .

Each object is unique

All cars have things in common: all cars have wheels, an engine, a steering device, a gas pedal...

Despite having things in common, every car in the world is a **unique** object. Every car has its own unique existence.

so, how are cars like computer-programmed objects?

All cars have things in common . . .

all cars have wheels, an engine, a steering device, a gas pedal . . .

Despite having things in common, every car in the world is a unique object. Every car has its own unique existence.

Computer-Programmed Objects

data related to the object are stored inside the object

the data are only changed by means of methods which are available to the object

depending on the data values, an outside method may be called to affect some other object

Computer-Programmed Objects

- data related to the object are stored inside the object (the car's speed, fuel)

Computer-Programmed Objects

data related to the object are stored inside the object, and this is called “information hiding” more commonly known as Encapsulation.

Computer-Programmed Objects

the data are only changed by means of methods (functions) which are also available to the object (a car's speed is altered by the accelerator method)

Computer-Programmed Objects

the data are only changed by means of functions which are available to the object (these functions are known as *methods* and alter the object's private data)

Computer-Programmed Objects

depending on the data values, an outside function may be called (when the seat belt is not fastened, an alarm will sound)

Computer-Programmed Objects

depending on the data values, an outside function may be called, in OOP this is called "raising an event"

All objects belong to a class

All objects within a class have the same methods

What is a class?

If I ask you if you own a computer, you will know, just by hearing the word “computer” that I mean . . .

. . . a machine with at least a keyboard, screen, processor and storage.

keyboard, screen, processor and storage. . .

these are some of the things that all computers have in common.

This is my definition of a generic computer and specifies for me the *Class* of 'Computer'.

Try to think of a class as being a description of an object, *but not the object itself* . . .

. . . a bit like the difference between a data type, and the data value...

. . . a bit like the difference between a chocolate cookie cutter and a chocolate cookie.

A class is a *template or blueprint* which can generate an object when called upon to do so.

All of the objects of a particular class have the features specified by that class.

You can create as many classes as you wish.

You can create as many objects of a class as you wish.

Because real-world objects can be mimicked by computer-programmed objects . . .

. . . we can create computer simulations which can, for example, teach a student pilot how to fly a 747 plane, but without any risk.

In fact, the very first OOP language was called *Simula* and it was designed to produce simulations

Objects belong to a Class

Consider a class as a template or blueprint of an object. The class contains all the information required to generate the object.

Objects have properties and functions

Objects have properties (attributes)- like a name, size, color, position.

Objects also can perform functions (methods) - like move, make noise, change direction, speed up, slow down.

Python 3 Classes

You can instantiate an object of a class by simply assigning the class name followed by any required arguments in the parentheses...

For instance, imagine we already have defined a class named "Person". Here is how to make two Person objects:

```
person1 = Person('Anne Dawson')
person2 = Person('Tom Lee')
```

Using Objects

Once we have created an object we can then access the attributes of the object by a notation such as:

```
print(person1.name)
```

Using Objects

Once we have declared an object we can then access the methods of the object by a notation such as, for example:

```
person1.moveleft()
person1.speak()
```

Classes

The Python language comes with a set of classes already written and ready for you to use.

You can use these classes, and you can write your own classes from which you can generate objects.

Just like there are predefined functions and user-defined functions, there are predefined classes and user-defined classes.

In OOP, programs are designed by identifying classes of objects, and by understanding the relationships between the objects.

End of Python3_Intro_OOP.odp

Last updated: Sunday 9th July 2017, 9:15 PT, AD