COMP10020
Introduction to Programming II
Object Oriented Programming

Dr. Brian Mac Namee
brian.macnamee@ucd.ie
School of Computer Science
University College Dublin

References

Some of the material in this lecture is based on "Gentle Object Oriented Programming in Python by Nicholas H.Tollervey & Naomi Ceder

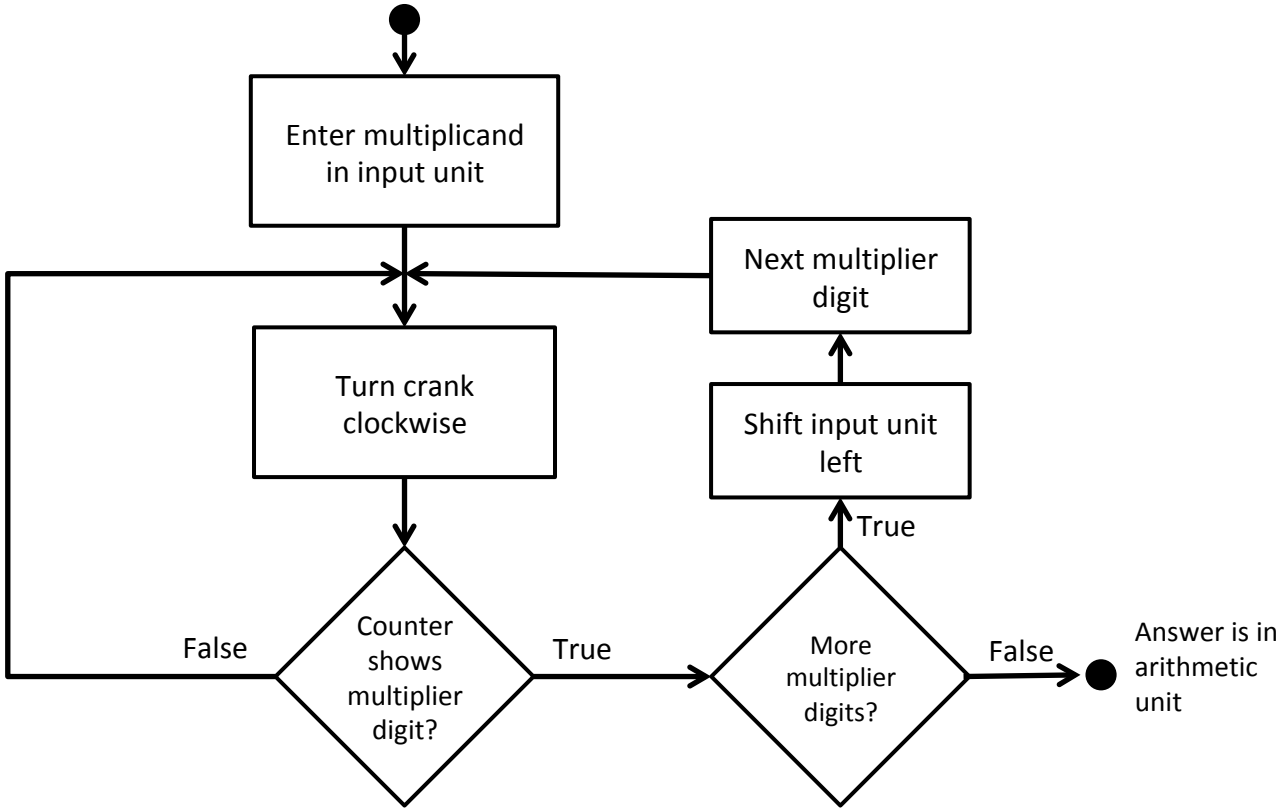
http://ntoll.org/static/presentations/oopy/index.html

WHY OBJECT ORIENTED PROGRAMMING?


UCD
DUBLIN
an coláiste ollscoile baile átha cliath
UNIVERSITY COLLEGE DUBLIN

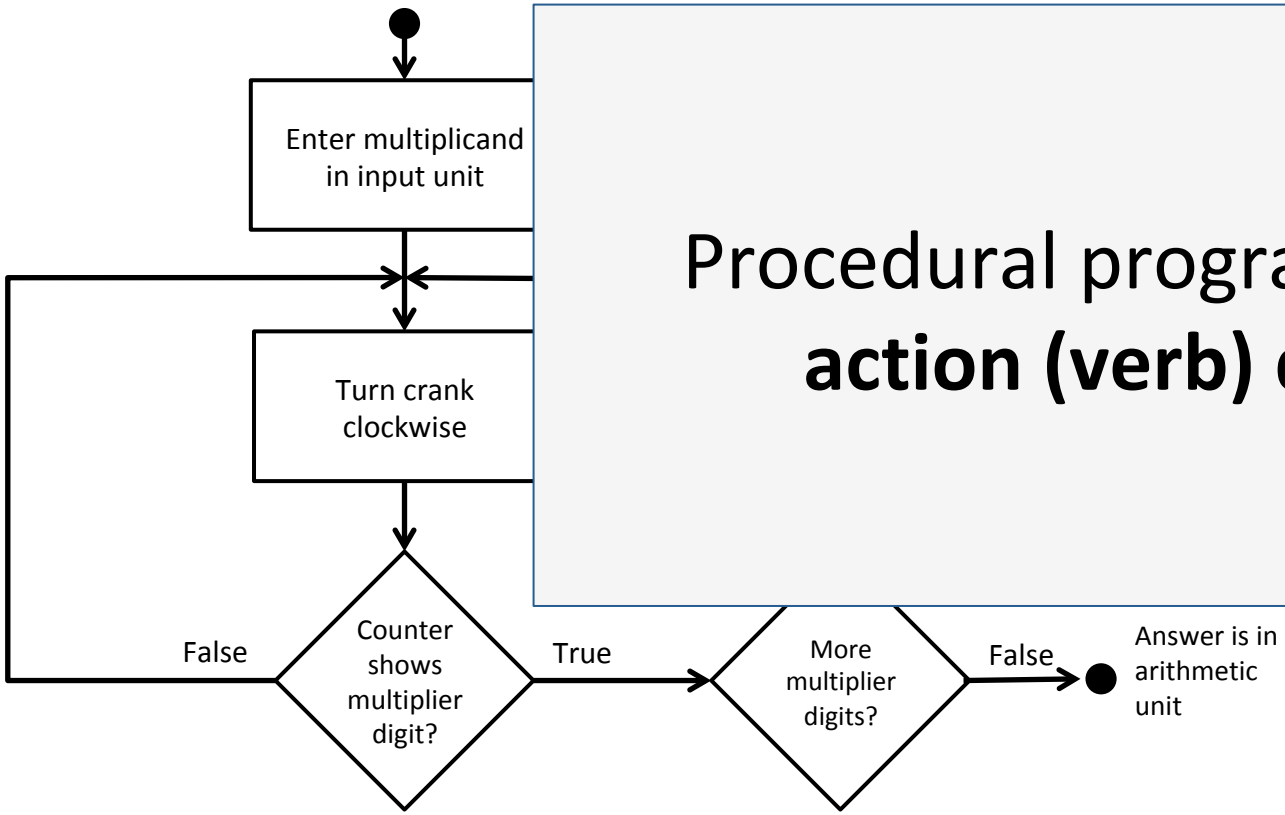
Procedural Programming

The programming we have looked at so far can be named **procedural programming**



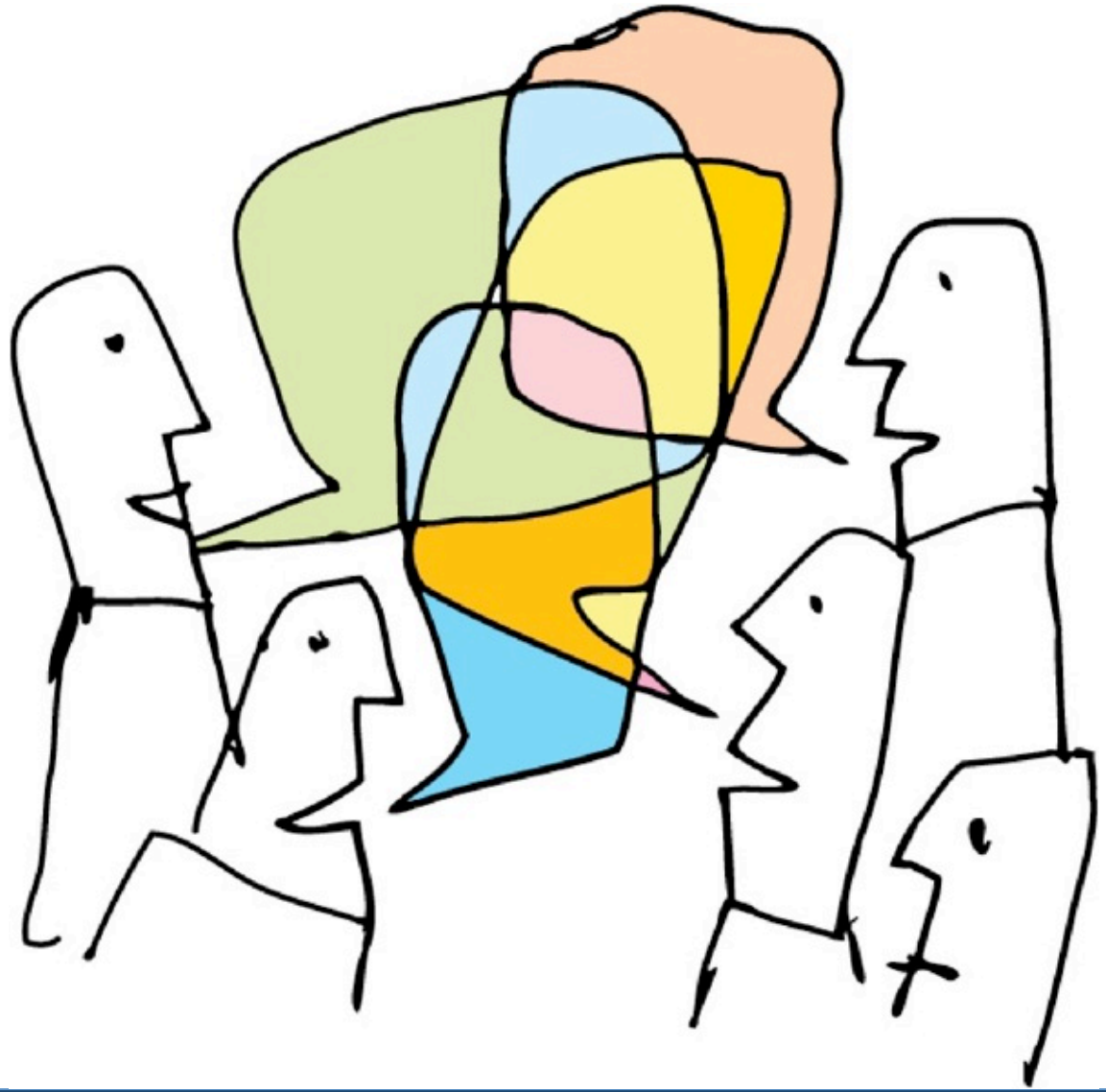
Procedural Programming

The programming we have looked at so far can be named **procedural programming**



Procedural programming is **action (verb) driven**

We Model the World with Language



Language Isn't Just Verbs!

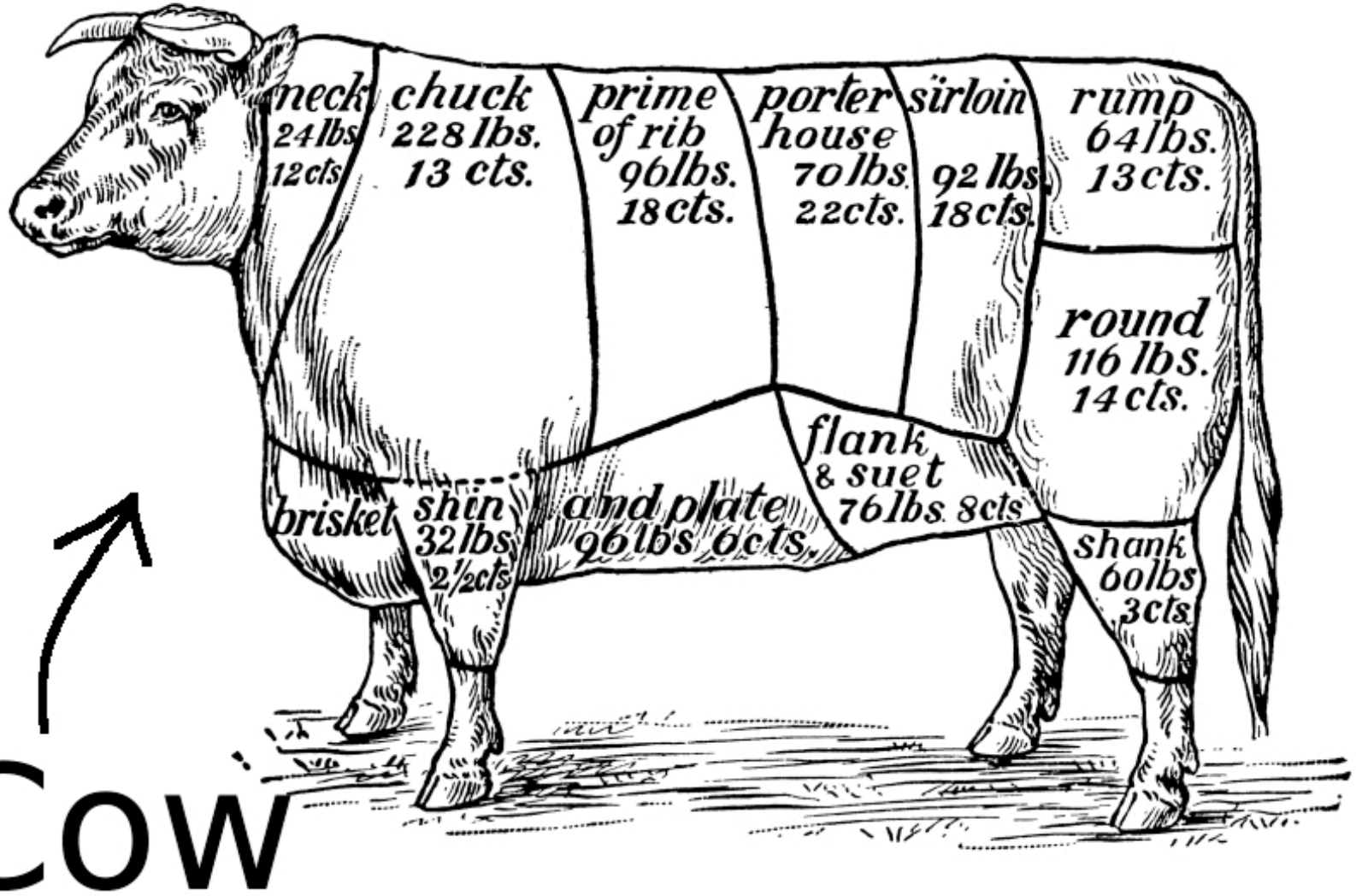
Language includes:

- Naming types of things (cats, dogs, cows)
- Naming specific things (Garfield, Snoopy, Buttercup)
- Describing actions (run, walk, smile, love, hate)
- Attributing qualities (red, furry, hot, scary)

Nouns, proper nouns, verbs, and adjectives

OBJECT ORIENTED BASICS

A **Class** Defines a Type of Thing



An **Object** is an Instance of a Class

A **Method** Specifies a Behaviour of an Object

An **Attribute** Defines a Characteristic of an Object

Back To Language

Class = Noun

Object = Proper noun

Method = Verb

Attribute = Adjective

(This is not entirely accurate but is close enough for today)

OBJECT ORIENTED PROGRAMMING IN PYTHON

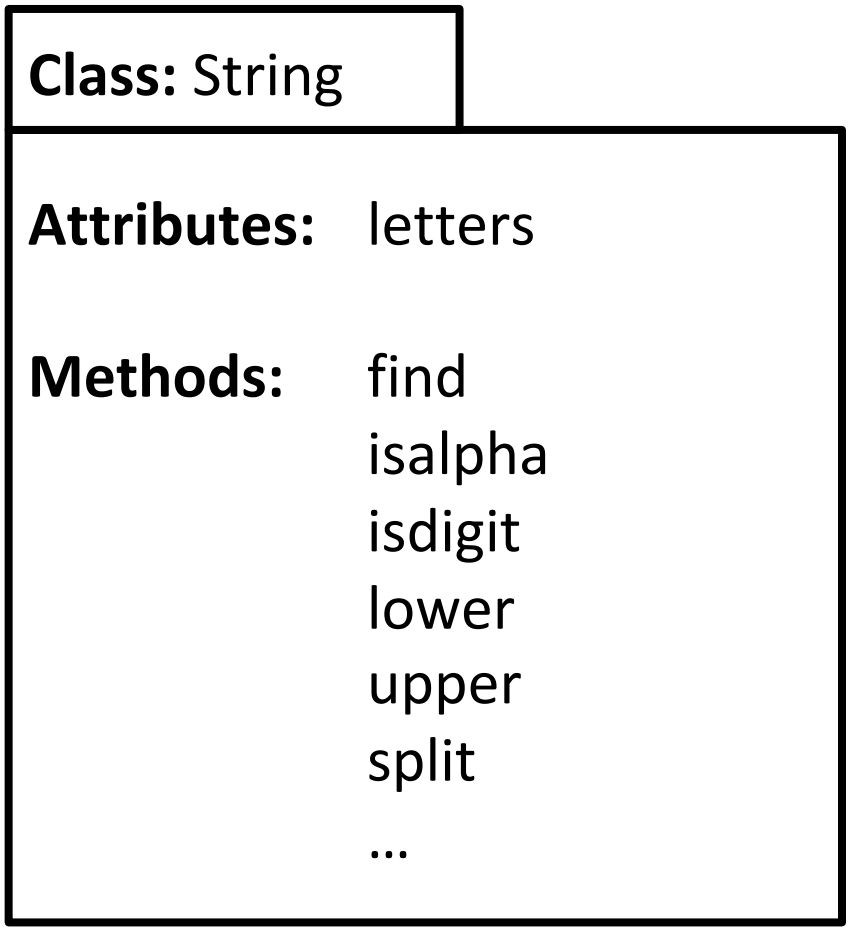
OOP in Python

We have already been using lots of objects in Python

Lists, dictionaries, strings etc are all objects

OOP in Python

Let's think about a string object



OOP in Python

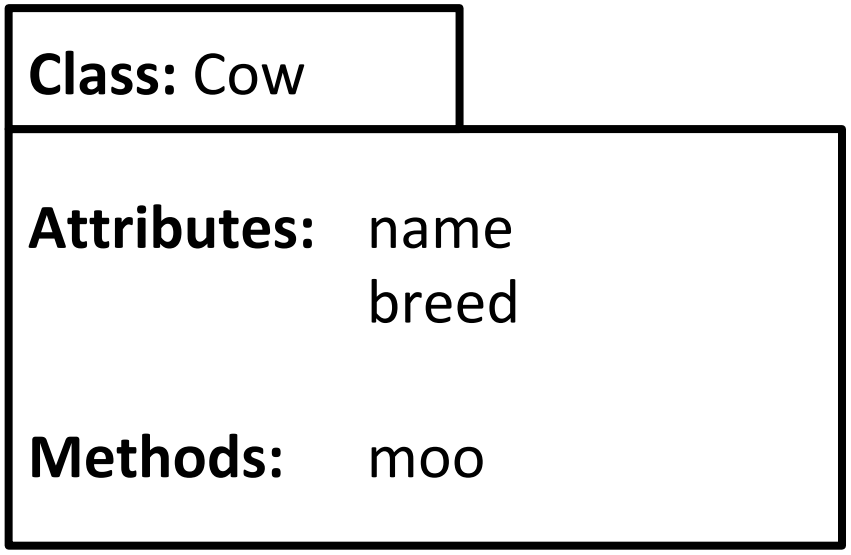
We can instantiate string objects and then perform methods on them

```
allLetters = "The quick brown fox jumped over the lazy dog"
print(allLetters.isalpha())
print(allLetters.isdigit())

words = allLetters.split()
print(words)
```

Defining a Class

The first step in defining a class is to define its attributes and methods



Then we can write Python code to define it

A Simple Pythonic Cow

```
class Cow:

    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def moo(self, message):
        print self.name +  " says " + message
```

A Simple Pythonic Cow

```
class Cow:

    def __init__(self, name, bree
        self.name = name
        self.breed = breed

    def moo(self, message):
        print self.name +  " says " + message
```

Use the **class** keyword to start a class definition
We can name a class anything we like – in this case Cow

A Simple Pythonic Cow

```
class Cow:

    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def moo(self, message):
        print self.name +  " says " + message
```

The **__init__** function is a special function called a **constructor** that is used to instantiate objects

A Simple Pythonic Cow

```
class Cow:

    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def moo(self, message):
        print self.name +  " says " + message
```

The parameters passed to the constructor are the pieces of information needed to instantiate a new object

A Simple Pythonic Cow

```
class Cow:

    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def moo(self, message):
        print self.name +  " says " + message
```

We always include the keyword **self** as a parameter to all class methods – this is just a bit of peculiar Python syntax

A Simple Pythonic Cow

```
class Cow:

    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def moo(self, message):
        print self.name +  " says " + message
```

Inside the constructor we declare the **instance attributes** of the class – the data that objects of the class contain

A Simple Pythonic Cow

```
class Cow:

    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def moo(self, message):
        print self.name +  " says " + message
```

We can then define the methods of the class

A Simple Pythonic Cow

```
class Cow:

    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def moo(self, message):
        print self.name +  " says " + message
```

We always use the **self** parameter

A Simple Pythonic Cow

```
class Cow:

    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def moo(self, message):
        print self.name +  " says " + message
```

To access instance attributes we use the **self** keyword

SUMMARY

Summary

In diving deeper into OO programming we will look at the four major principles:

- **Encapsulation** <- looked at today
- Composition
- Inheritance
- Polymorphism