Logistics

Reading 5 has been assigned. (Due next Sunday)

Homework 5 has been assigned. (Due next Sunday)

There is a quiz in your lab sections this week.

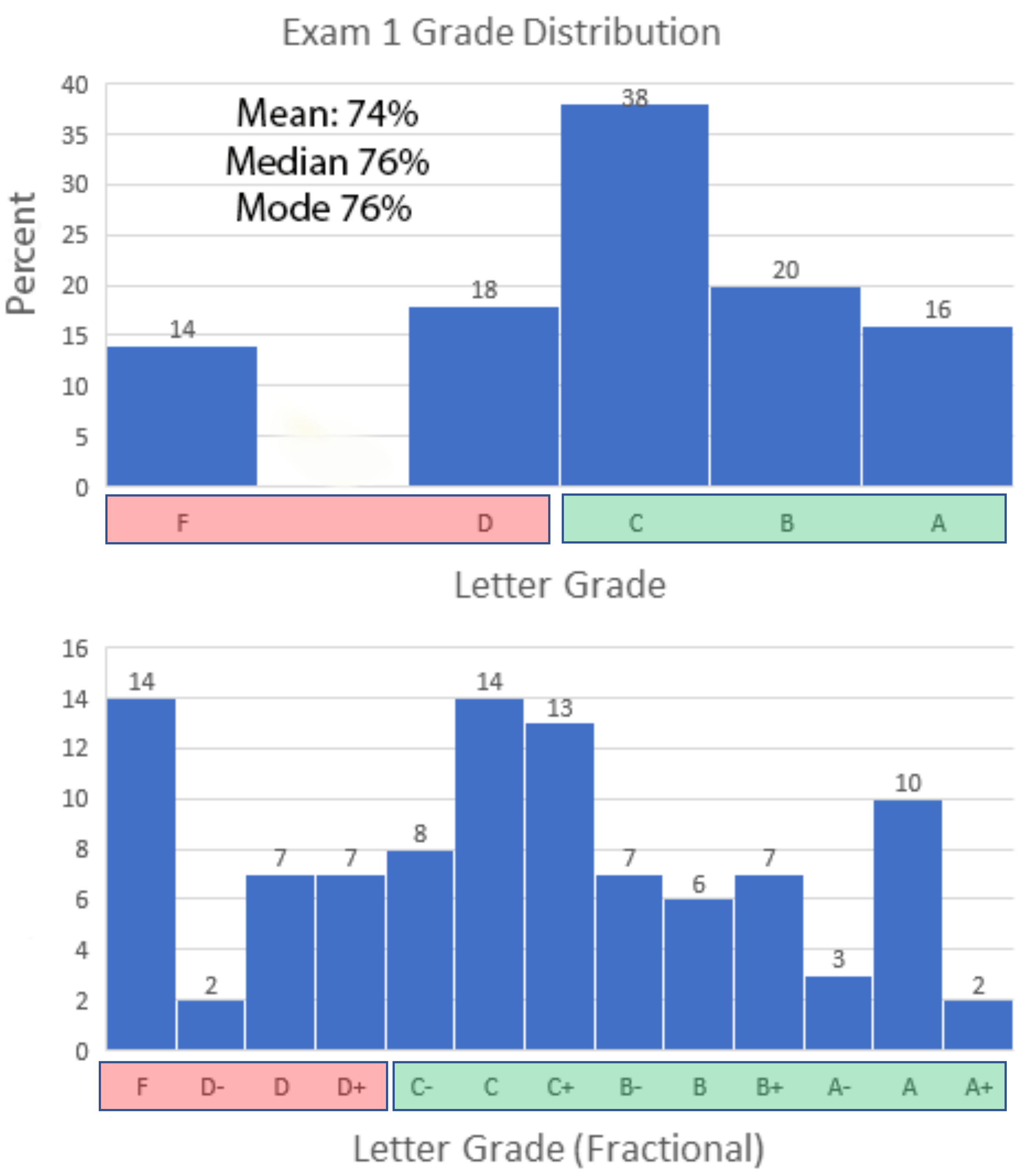
Exam 1 grade distribution and solution key have been posted.

Exam Grade distribution

34% of the exams were in the “Good” or “Excellent” range.

36% of the exams were in the “satisfactory” range.

30% of the exams were in the “not satisfactory “or below range.

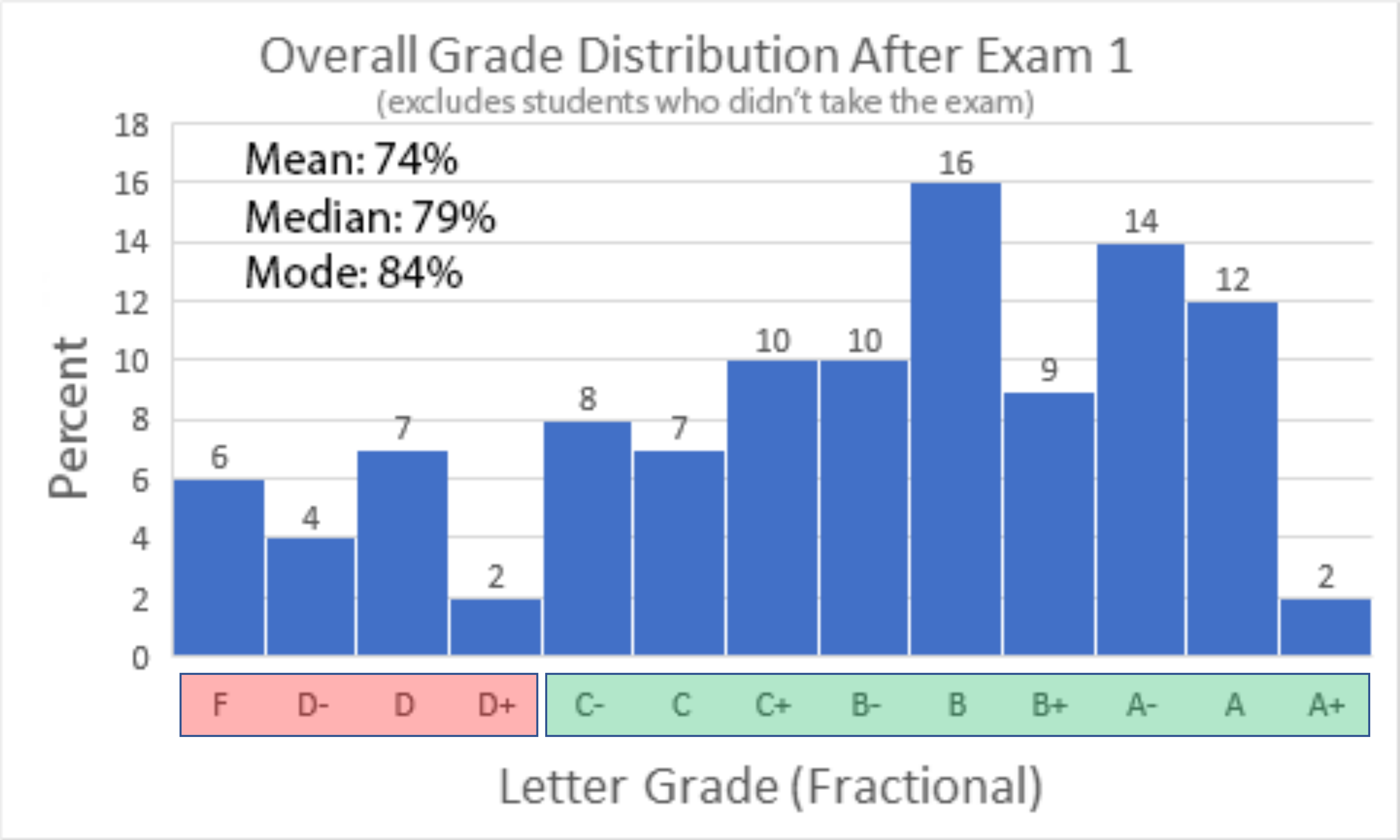

Exam 1 Grade Distribution

Mean: 74%
Median 76%
Mode 76%

Overall Grades

82% of the class has a grade of “satisfactory” or above.

Congratulations!



Please come to office hours if you have a grade below C-. We are here to help you get into the green zone.

Name a profession or job...

Name tasks someone with that job does...

Name things someone with that job has to have or know...

Programming Styles

There are many styles of programming, for example:

- Procedural Programming (basically what you have learned so far)
- Functional Programming (everything is based on functions)
- Logic Programming (everything is based on Boolean statements)
- Object Oriented Programming (OOP) (everything is based on Objects)

- Python is a good first language to learn because it supports procedural, functional and object-oriented programming (OOP).
- Most programmers use a mix of all these styles.

Object-Oriented Programming

We want our programs to do tasks for us.

One way to organize our thinking and programs is to create pieces of code that have well-defined “jobs”.

That is how we have organized our society…

Why do we organise work in our society by job name?

Object Oriented Programming

The same reasons we organize our society into people with jobs
and professions apply to programs:

- We know exactly who to go to if we need something done, even if we know nothing about the person other than their job title.
- We have an idea that someone with a particular profession will be able to do certain things.
- The job may encompass things we have no idea of how to do.
- The person with the job is an expert. They might improve how they do things without us having to worry about the tasks they can do changing.

Object Oriented Programming

- One of the first applications of modern computing was modelling and simulation.
- Scientists soon realized that functions alone were insufficient to model systems intuitively
- If we are going to model a planet we would like to actually create a virtual planet, define how it behaves in our simulated universe, and then just observe it.

Object Oriented Programming

- Programmers quickly realized that the idea of creating virtual “things” made software engineering simpler to think about.
- If we create within our programs agents and objects then we can assign duties and tasks to them.
- This is really just another way applying decomposition to our software.
- Break up the problem to be solved into logical parts and assign each part to an object.

Object Oriented Programming

- Even engineers are social animals - we evolved to think about the world in terms of agents and objects (not recursion).
- In many situations we solve large problems by delegation. That is we have workers who specialize in solving a particular problem.
- Those specialists have specific skills that they can apply to a specific class of problems.

Object Oriented Programming

- We can pattern software after a group of specialists at a company working on a problem.
- For example, there are two objects we have used – string and turtle.
- String is the name of an object who knows all about storing characters and answering questions about them.
- Turtle knows how to draw something on the screen, and perform operations like forward, turn left, etc

Object Oriented Programming

- Important: we don’t have to have any idea how turtle does its job. We just trust that it does.
- Just like we don’t question the US Mail about how our letter gets from here to Seattle.
- We only care that it arrives within certain tolerances – not how it got there.
- This is called abstraction, information hiding, and encapsulation and we like it!

Object Oriented Programming

- When we mail a letter all we have to worry about is following the post office procedure to ensure our letter gets to the right place.
- We have to know where to go, how to pay, the format expected for the destination address and return address, etc.
- In software this is called the interface.
- All objects have to have an interface that clearly defines how we can interact with the object.

Object Oriented Programming

Almost any problem can be broken up into objects.

- Objects are defined by three things:
    - Their state – this is the information they contain.
    - Their behaviour or capabilities – these are the member functions they have access to.
    - Their interface – the rules describing how they interact with other objects in the system.

Reasons for OOP

Abstraction

Encapsulation

Information hiding

Inheritance

Class: Object Types

- Like other OOP languages, Python uses classes to define objects
- A Python class specifies the type of an object.
- When you define a class you are specifying the attributes and behaviour of a new type.
  - ➢Classes have member variables and member functions (aka methods)
  - ➢ Behaviour is defined by member functions

Information Hiding

- The interface acts as a contract specifying how the object will behave – as long as the code fulfils the contract we don’t care how it works.
- Defining a class does not result in creation of an object.
- Declaring a variable of a class type creates an object. You can have many variables of the same type (class).
- This is called instantiation of the class

Information Hiding (cont.)

- This is good because it allows us to change the underlying code without forcing everyone who uses our objects to change their code.
- You can change the implementation and nobody cares! (as long as the interface is the same).
- We never have to worry if the US Post office decides to use a train instead of a truck, as long as the letter arrives on time. The interface remains the same.

Private vs. Public (note)

- If you are coming from another OOP language, Python does not have real support for private variables and functions.

Special Member Functions

- Constructors: called when a new object is created (instantiated).

Python Classes: Create a Virtual Dog!

```
class Dog:
    kind = 'canine'  # class variable shared by all instances
    self.sound = “Woof!”

    def __init__(self, name):   # Constructor
        self.name = name # instance variable unique to each instance

    def name(self):   # Member function
        return(self.name)

    def sound(self):   # Member function
        return(self.sound)
```

Python Classes: Create a Virtual Dog!

```
class Dog:
    kind = 'canine'  # class variable shared by all instances
    self.sound = “Woof!”

    def __init__(self, name):   # Constructor
        self.name = name # instance variable unique to each instance

    def name(self):   # Member function
        return(self.name)

    def sound(self):   # Member function
        return(self.sound)
```

Self: The name this object calls itself.

Python Classes: Create a Virtual Dog!

```
class Dog:
    kind = 'canine'  # class variable shared by all instances
    self.sound = “Woof!”

    def __init__(self, name):   # Constructor
        self.name = name # instance variable unique to each instance

    def name(self):   # Member function
        return(self.name)

    def sound(self):   # Member function
        return(self.sound)
```

Special function __init__

This is the constructor.

Python Classes: Create a Virtual Dog!

```
class Dog:
    kind = 'canine'  # class variable shared by all instances
    self.sound = “Woof!”

    def __init__(self, name):   # Constructor
        self.name = name # instance variable unique to each instance

    def name(self):   # Member function
        return(self.name)

    def sound(self):   # Member function
        return(self.sound)
```

If you don´t specify self
The member variable is shared
by all objects of type “Dog”.

The member variable, e.g.
“kind” is shared.

Python Classes: Create a Virtual Dog!

```
class Dog:
    kind = 'canine'  # class variable shared by all instances
    self.sound = “Woof!”

    def __init__(self, name):   # Constructor
        self.name = name # instance variable unique to each instance

    def name(self):   # Member function
        return(self.name)

    def sound(self):   # Member function
        return(self.sound)
```

Member functions that define what objects of type Dog can do.

In this example Dogs can give you their “name” and they can make a “sound”.

Python Classes: Create a Virtual Dog!

```
class Dog:
        kind = 'canine' # class variable shared by all instances
        self.sound = “Woof!”

        def __init__(self, name):  # Constructor
                self.name = name # instance variable unique to each instance

        def sound(self):  # Member function
                return(self.name)

        def sound(self):  # Member function
                return(self.sound)
```

Instantiating the class into a dog object.

Pass in its name as an argument.

```
mydog = dog(“Fido”)
print( mydog.name() + “ says ”+ mydog.sound() )

Fido says Woof!
```

Python Classes: Create a Virtual Dog!

```
class Dog:
         kind = 'canine' # class variable shared by all instances
         self.sound = “Woof!”

         def __init__(self, name):  # Constructor
                  self.name = name # instance variable unique to each instance

         def sound(self):  # Member function
                  return(self.name)

         def sound(self):  # Member function
                  return(self.sound)
```

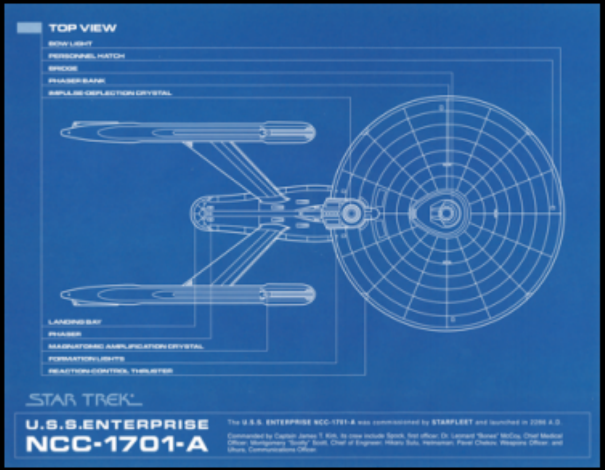
Use the member functions...

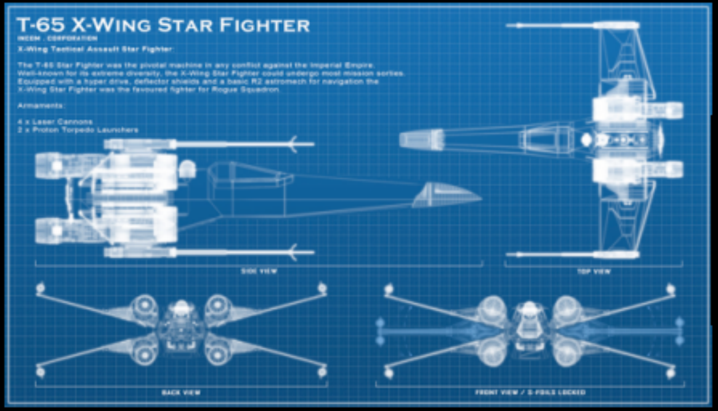
```
mydog = dog(“Fido”)
print( mydog.name() + “ says ”+ mydog.sound() )

Fido says Woof!
```

Classes vs Objects

Classes are like the job description

The object is the person hired to do the job.



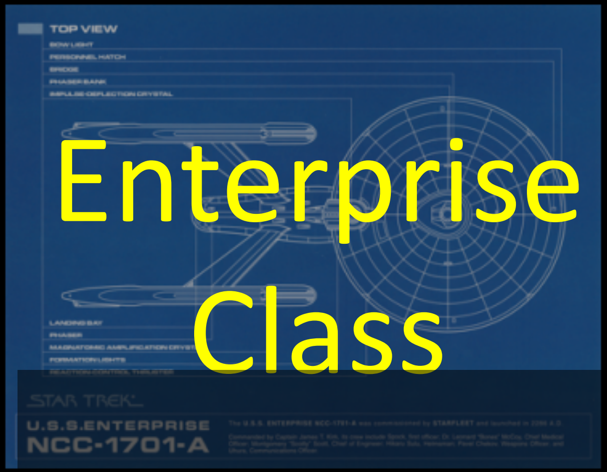

T-65 X-WING STAR FIGHTER
INCOM . CORPORATION
X-Wing Tactical Assault Star Fighter:
The T-65 Star Fighter was the pivotal machine in any conflict against the Imperial Empire. Well-known for its extreme diversity, the X-Wing Star Fighter could undergo most mission sorties. Equipped with a hyper drive, deflector shields and a basic R2 astromech for navigation the X-Wing Star Fighter was the favoured fighter for Rogue Squadron.
Armaments:
4 x Laser Cannons
2 x Proton Torpedo Launchers
SIDE VIEW
TOP VIEW
BACK VIEW
FRONT VIEW / S-FOILS LOCKED

- Imagine when you are writing a class that it is a blueprint.
- Instantiating a class is building the object described by the blueprint.

Classes vs Objects

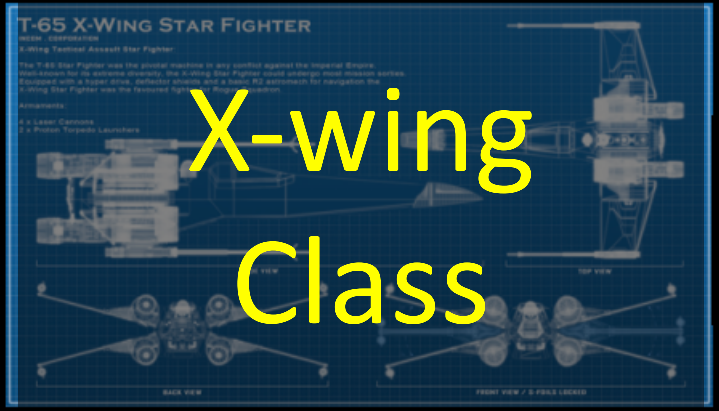
Classes are like the job description

The object is the person hired to do the job.



Enterprise Class

Enterprise Objects

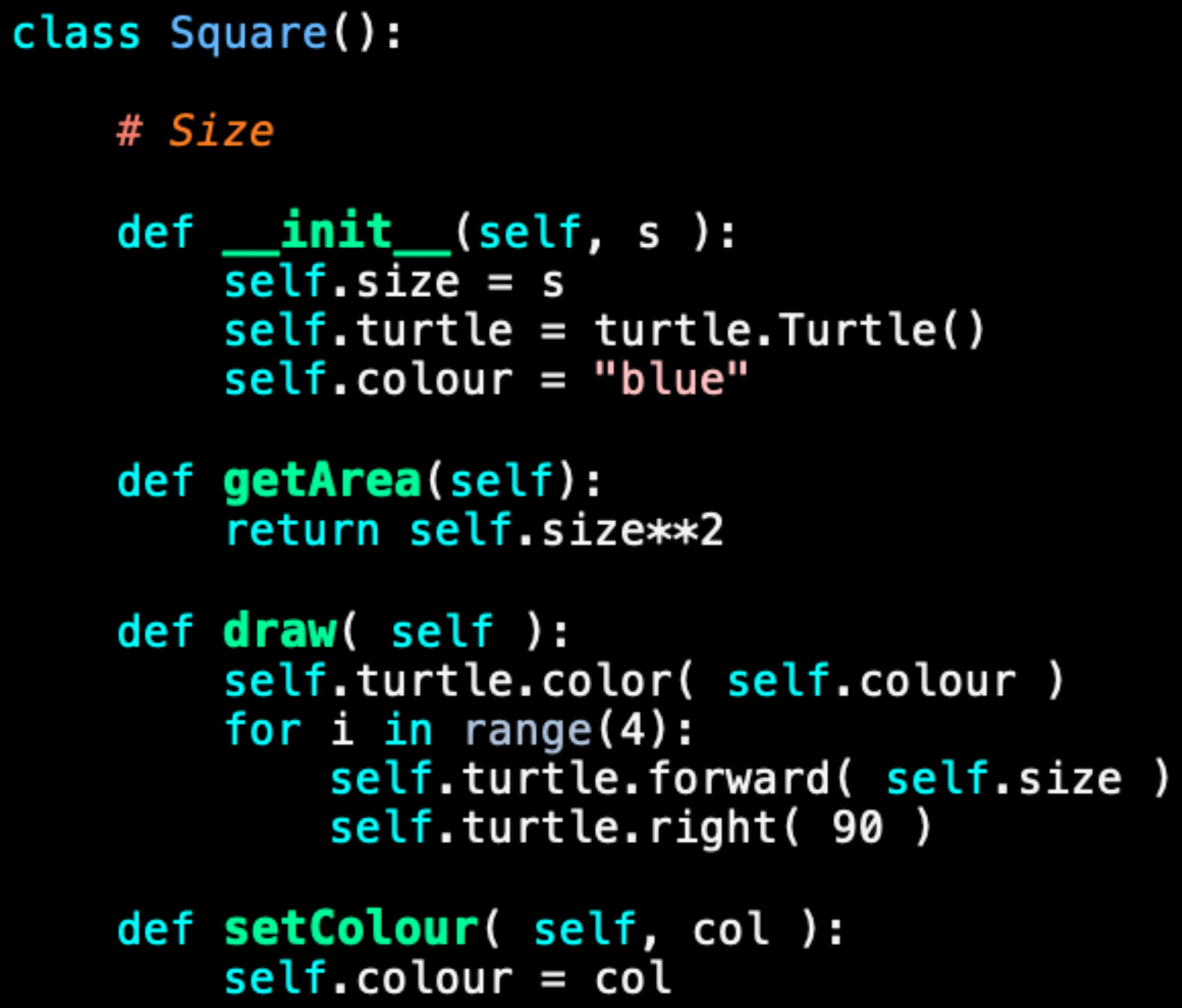


X-wing Class

X-wingObjects

- Imagine when you are writing a class that it is a blueprint.
- Instantiating a class is building the object described by the blueprint.

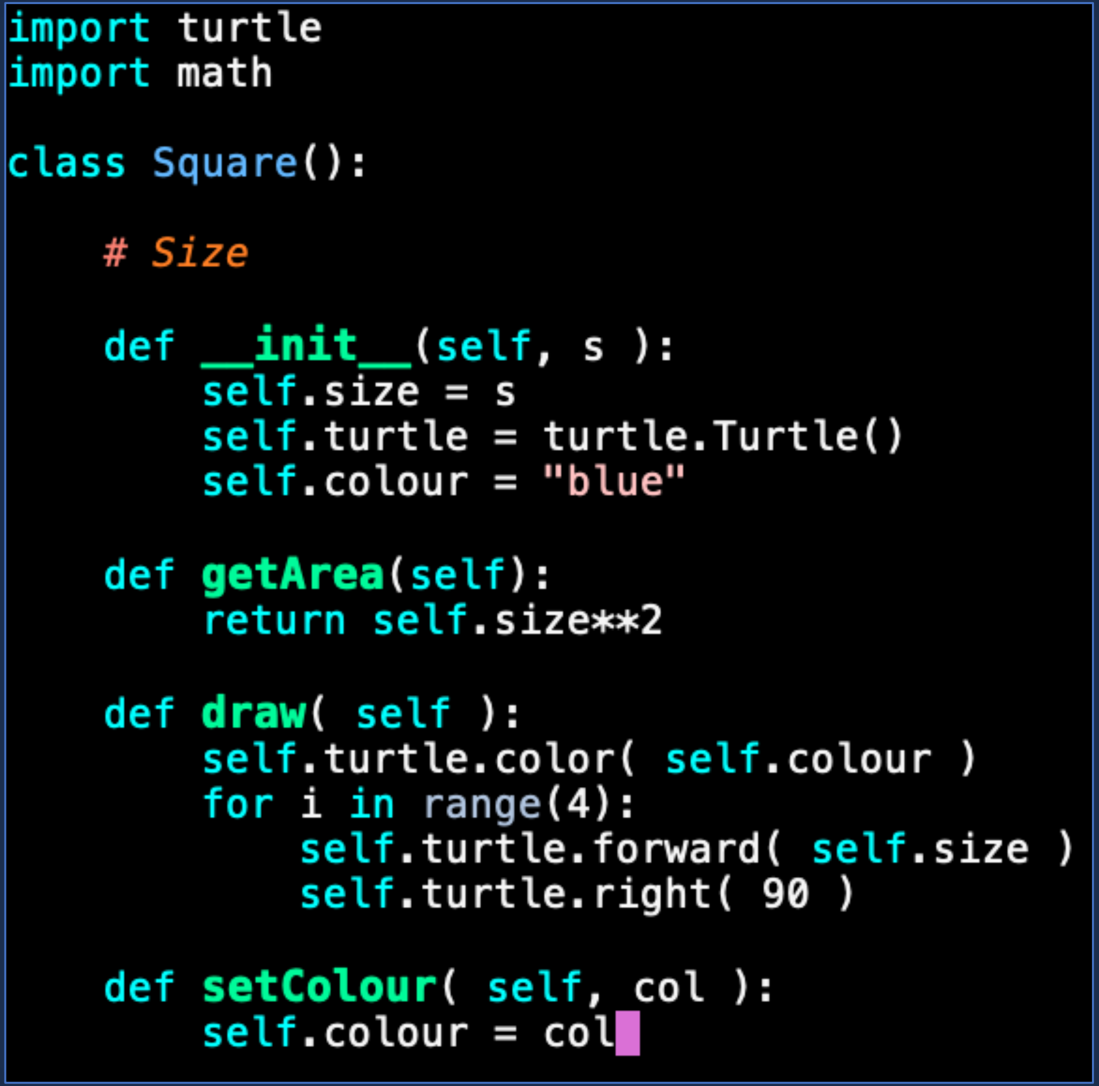
Shapes Example – Defining a “Square” class



```
class Square():

    # Size

    def __init__(self, s ):
        self.size = s
        self.turtle = turtle.Turtle()
        self.colour = "blue"

    def getArea(self):
        return self.size**2

    def draw( self ):
        self.turtle.color( self.colour )
        for i in range(4):
            self.turtle.forward( self.size )
            self.turtle.right( 90 )

    def setColour( self, col ):
        self.colour = col
```

Shapes Example – Defining a “Square” class

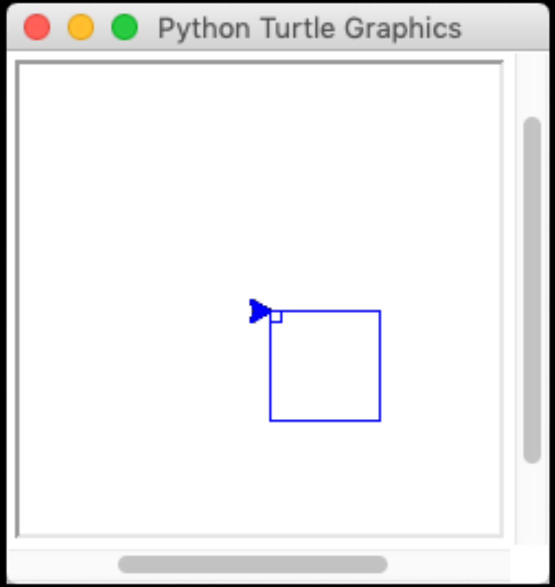
In shapes.py



```
import turtle
import math

class Square():

    # Size

    def __init__(self, s ):
        self.size = s
        self.turtle = turtle.Turtle()
        self.colour = "blue"

    def getArea(self):
        return self.size**2

    def draw( self ):
        self.turtle.color( self.colour )
        for i in range(4):
            self.turtle.forward( self.size )
            self.turtle.right( 90 )

    def setColour( self, col ):
        self.colour = col
```

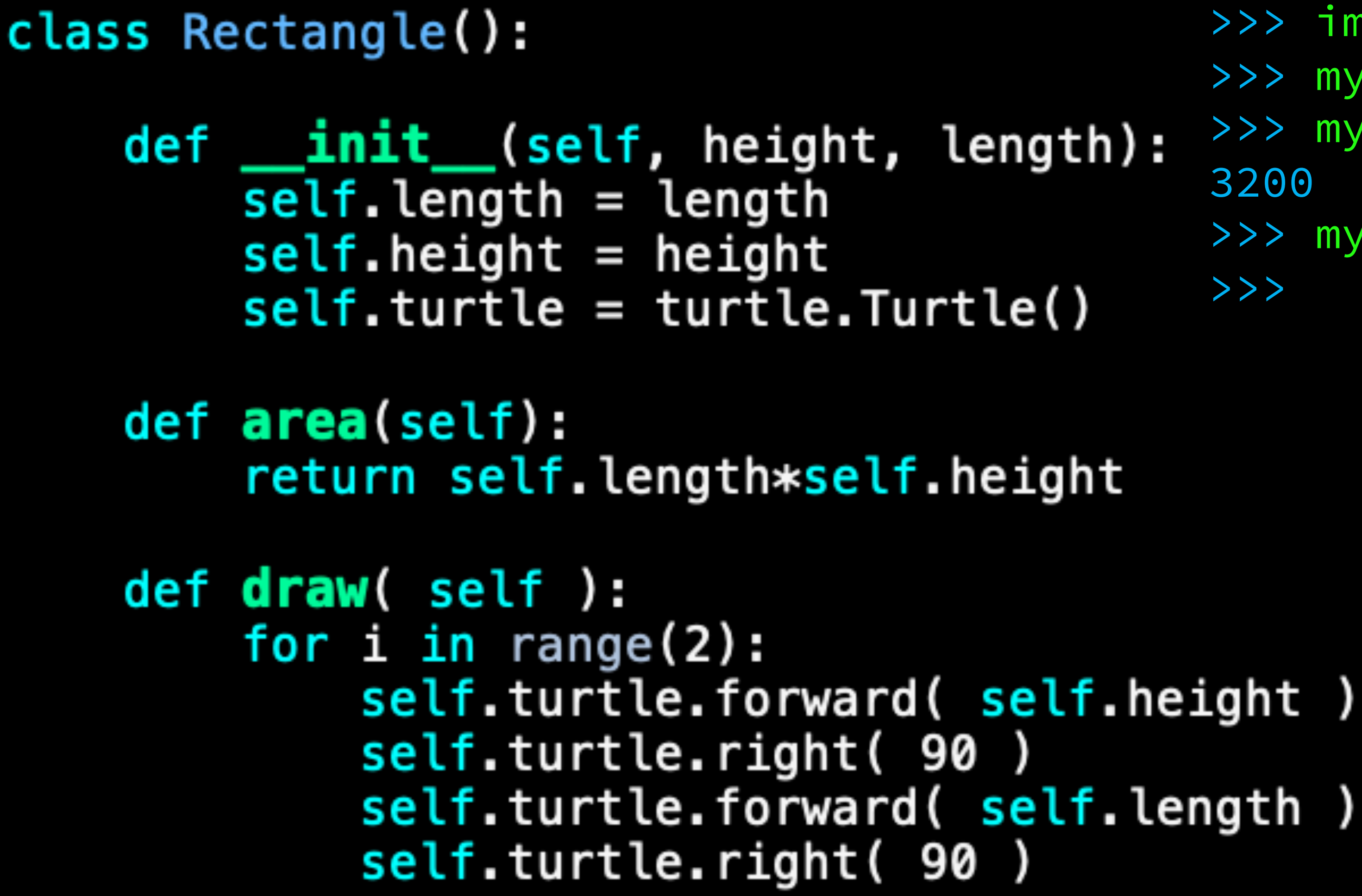
In python3 interpreter

```
>>> import shapes
>>> my_square = shapes.Square(50)
>>> my_square.getArea()
2500
>>> my_square.draw()
>>>
```



Shapes Example – Defining a “Rectangle” class

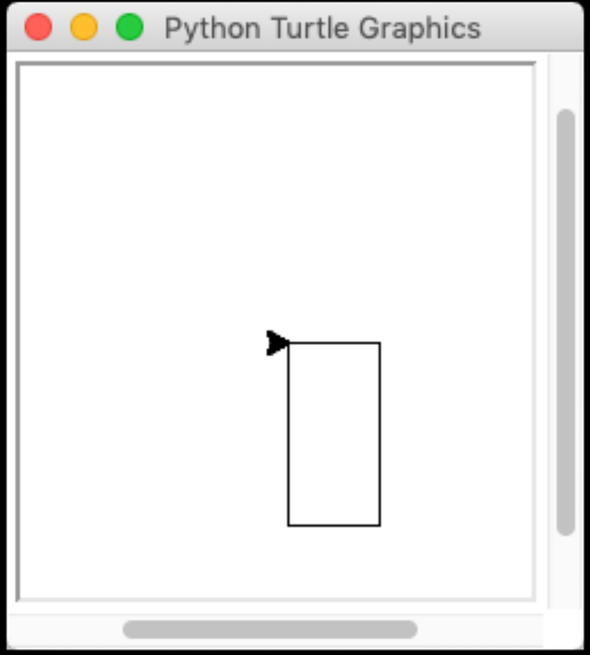
In shapes.py



```
class Rectangle():

    def __init__(self, height, length):
        self.length = length
        self.height = height
        self.turtle = turtle.Turtle()

    def area(self):
        return self.length*self.height

    def draw( self ):
        for i in range(2):
            self.turtle.forward( self.height )
            self.turtle.right( 90 )
            self.turtle.forward( self.length )
            self.turtle.right( 90 )
```

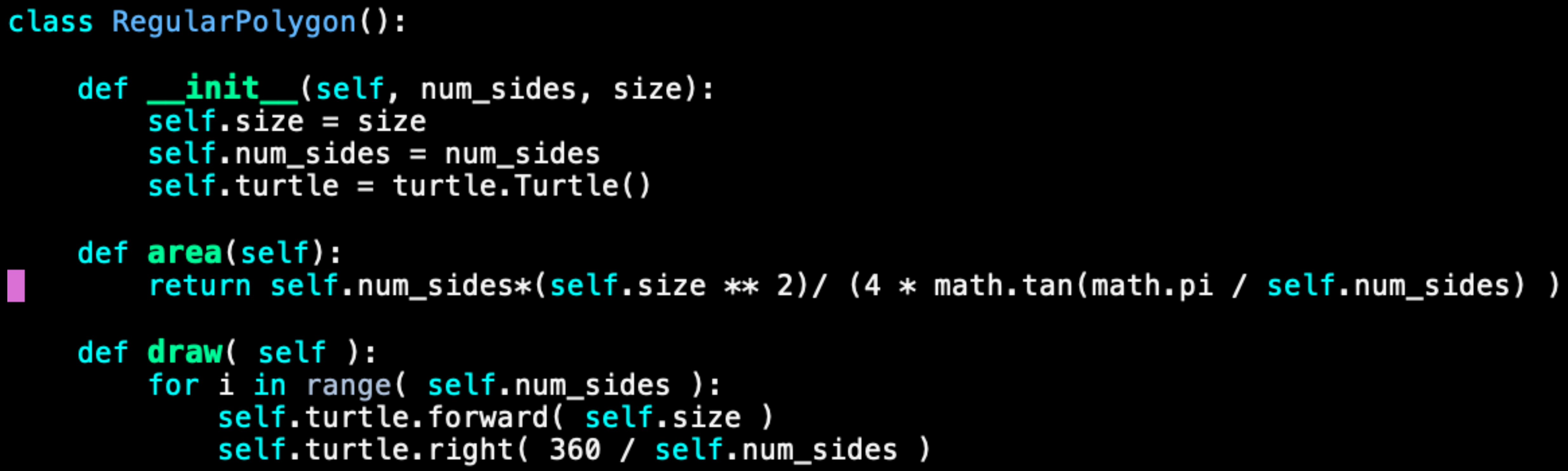
In python3 interpreter

```
>>> import shapes
>>> my_rectangle = shapes.Rectangle(40,80)
>>> my_rectangle.area()
3200
>>> my_rectangle.draw()
>>>
```



Shapes Example – Defining a “RegularPolygon” class

In shapes.py

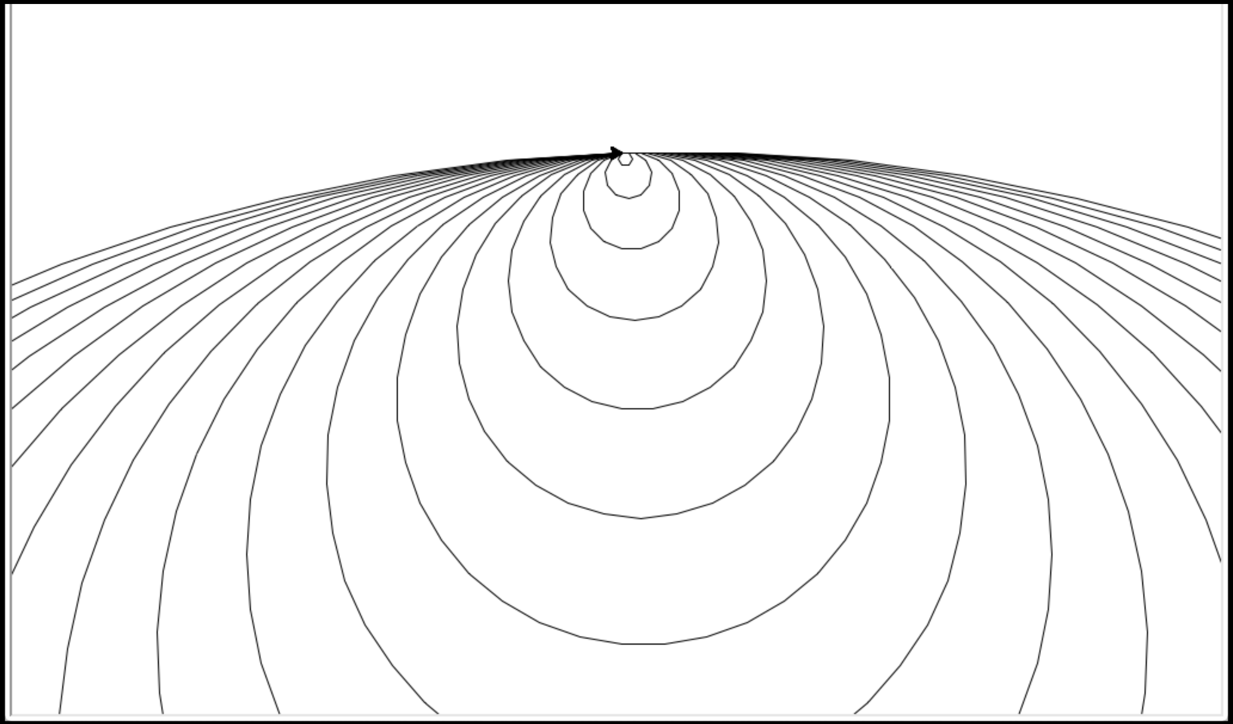


```
class RegularPolygon():

    def __init__(self, num_sides, size):
        self.size = size
        self.num_sides = num_sides
        self.turtle = turtle.Turtle()

    def area(self):
        return self.num_sides*(self.size ** 2)/ (4 * math.tan(math.pi / self.num_sides) )

    def draw( self ):
        for i in range( self.num_sides ):
            self.turtle.forward( self.size )
            self.turtle.right( 360 / self.num_sides )
```

Shapes Example – Defining a “RegularPolygon” class

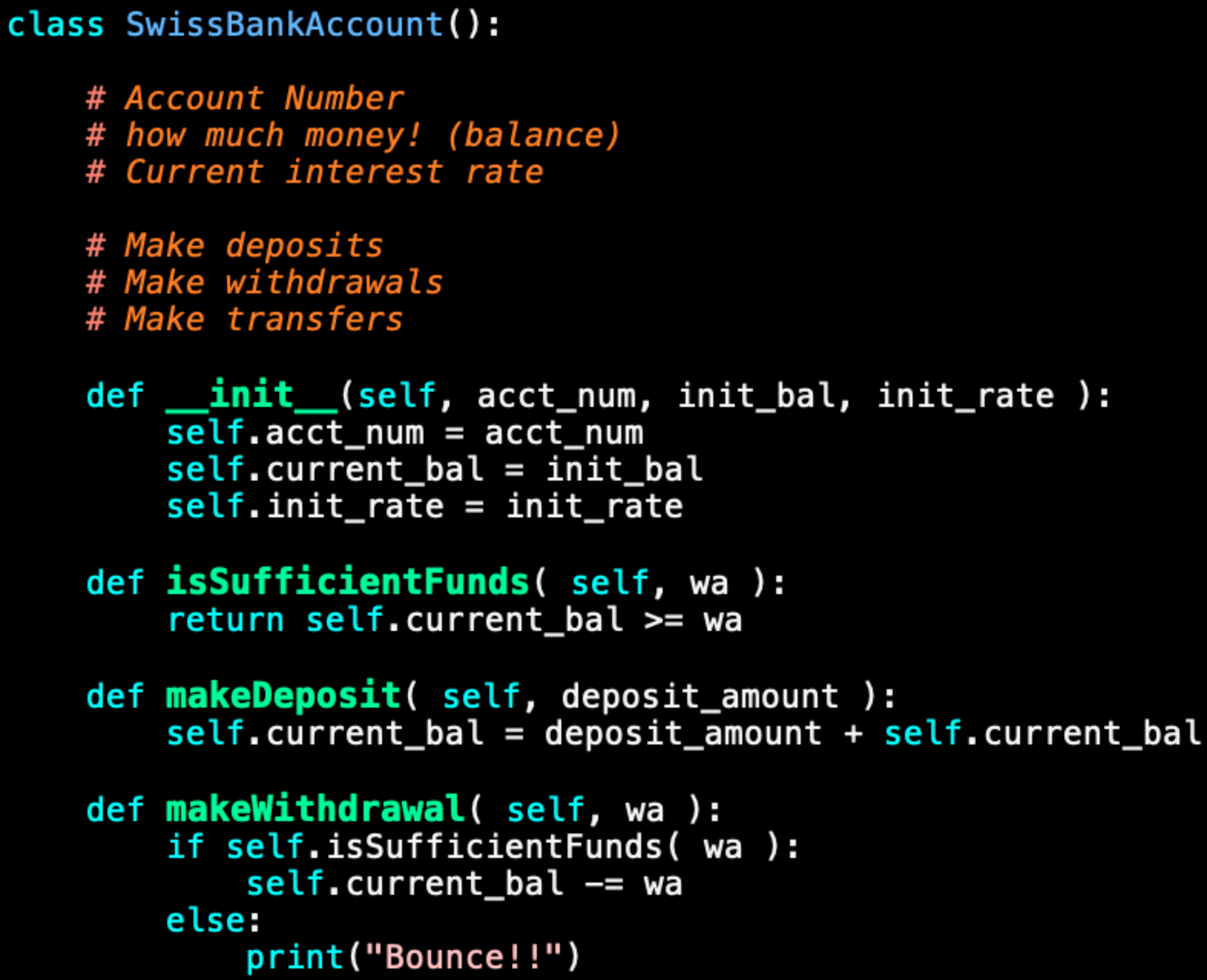
In python3 interpreter

```
>>> import shapes
>>> some_polys = []
>>> for i in range(1, 100, 5):
...     some_polys.append(shapes.RegularPolygon(i, i))
...
>>> for i in some_polys:
...     i.draw()
...
>>>
```



Bank Account Example

In bankaccount.py



```
class SwissBankAccount():

    # Account Number
    # how much money! (balance)
    # Current interest rate

    # Make deposits
    # Make withdrawals
    # Make transfers

    def __init__(self, acct_num, init_bal, init_rate ):
        self.acct_num = acct_num
        self.current_bal = init_bal
        self.init_rate = init_rate

    def isSufficientFunds( self, wa ):
        return self.current_bal >= wa

    def makeDeposit( self, deposit_amount ):
        self.current_bal = deposit_amount + self.current_bal

    def makeWithdrawal( self, wa ):
        if self.isSufficientFunds( wa ):
            self.current_bal -= wa
        else:
            print("Bounce!!")
```

Bank Account Example

```
>>> import bankaccount
>>> x = bankaccount.SwissBankAccount(10002034,100, 0.0012)
>>> x.current_bal
100
>>> x.makeWithdrawal(20)
>>> x.current_bal
80
>>> x.makeDeposit(50)
>>> x.current_bal
130
>>> x.makeWithdrawal(120)
>>> x.makeWithdrawal(120)
You have no money!!
>>> x.current_bal
10
>>> x.makeWithdrawal(5)
>>> x.current_bal
5
```