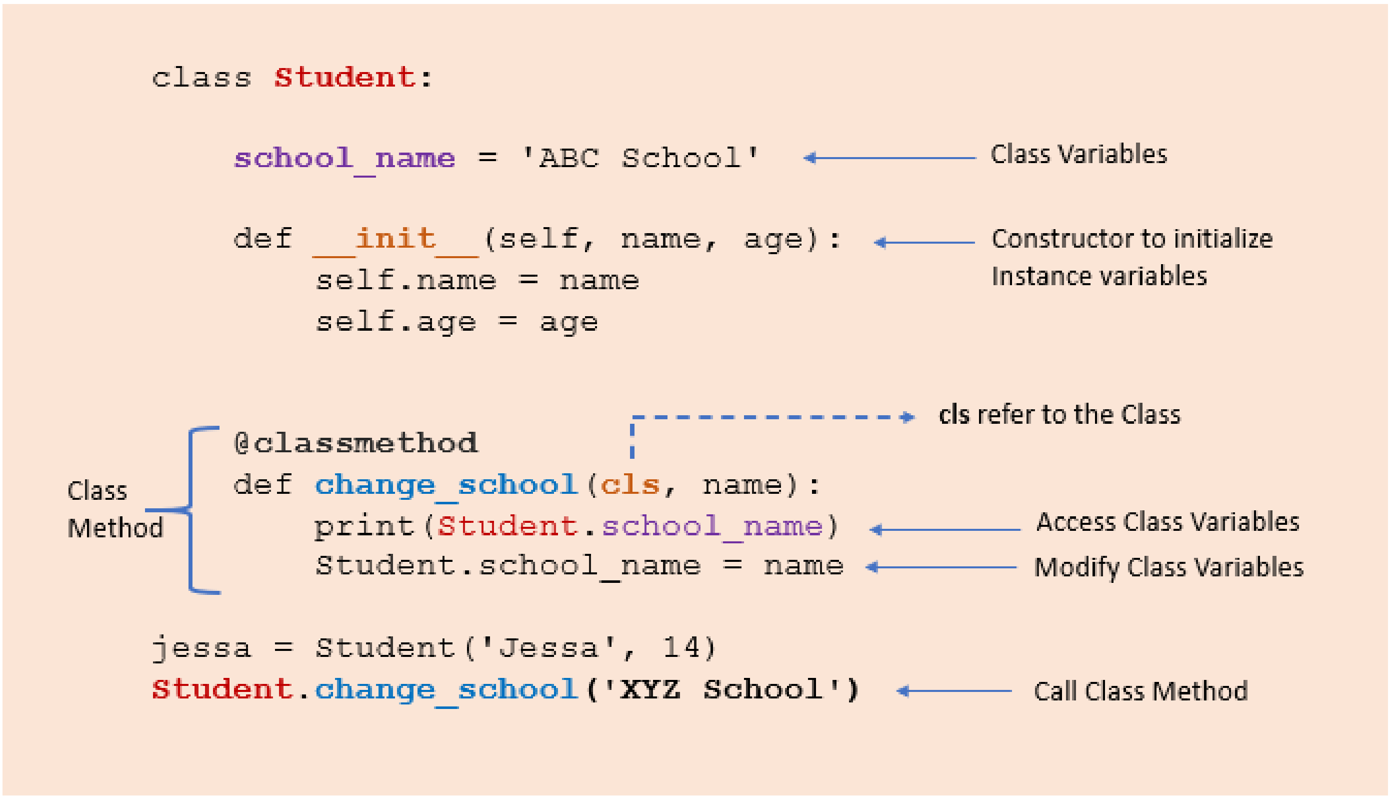
Python Tutorial

Created by Mustafa Germec

12. Classes and Objects in Python

- Python is an **object-oriented programming language**.
- Unlike procedure-oriented programming, where the main emphasis is on functions, object-oriented programming stresses on objects.
- An **object** is simply a collection of data (variables) and methods (functions) that act on those data.
- Similarly, a **class** is a blueprint for that object.
- Like function definitions begin with the **def** keyword in Python, class definitions begin with a **class** keyword.
- The first string inside the class is called **docstring** and has a brief description of the class.
- Although not mandatory, this is highly recommended.



```
class Student:

    school_name = 'ABC School'

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

    @classmethod
    def change_school(cls, name):
        print(Student.school_name)
        Student.school_name = name

jessa = Student('Jessa', 14)
Student.change_school('XYZ School')
```

Create a class

In [40]:

```
class Data:
    num = 3.14

print(Data)
```

```
<class '__main__.Data'>
```

Create an object

In [41]:

```
class Data:
    num = 3.14

var = Data()
print(var.num)
```

```
3.14
```

Function init()

In [43]:

```
class Data:
    def __init__(self, euler_number, pi_number, golden_ratio):
        self.euler_number = euler_number
        self.pi_number = pi_number
        self.golden_ratio = golden_ratio

val = Data(2.718, 3.14, 1.618)

print(val.euler_number)
print(val.golden_ratio)
print(val.pi_number)
```

```
2.718
1.618
3.14
```

Methods

In [45]:

```
class Data:
    def __init__(self, euler_number, pi_number, golden_ratio):
        self.euler_number = euler_number
        self.pi_number = pi_number
        self.golden_ratio = golden_ratio
    def msg_function(self):
        print("The euler number is", self.euler_number)
        print("The golden ratio is", self.golden_ratio)
        print("The pi number is", self.pi_number)

val = Data(2.718, 3.14, 1.618)
val.msg_function()
```

```
The euler number is 2.718
The golden ratio is 1.618
The pi number is 3.14
```

Self parameter

- The **self parameter** is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named **self**, you can call it whatever you like, but it has to be the **first parameter** of any function in the **class**.
- Check the following example:

In [46]:

```
"""
The following codes are the same as the above codes under the title 'Methods'.
You see that the output is the same, but this codes contain 'classFirstParameter' instead of 'self'.
"""
class Data:
    def __init__(classFirstParameter, euler_number, pi_number, golden_ratio):
        classFirstParameter.euler_number = euler_number
        classFirstParameter.pi_number = pi_number
        classFirstParameter.golden_ratio = golden_ratio

    def msg_function(classFirstParameter):
        print("The euler number is", classFirstParameter.euler_number)
        print("The golden ratio is", classFirstParameter.golden_ratio)
        print("The pi number is", classFirstParameter.pi_number)

val = Data(2.718, 3.14, 1.618)
val.msg_function()
```

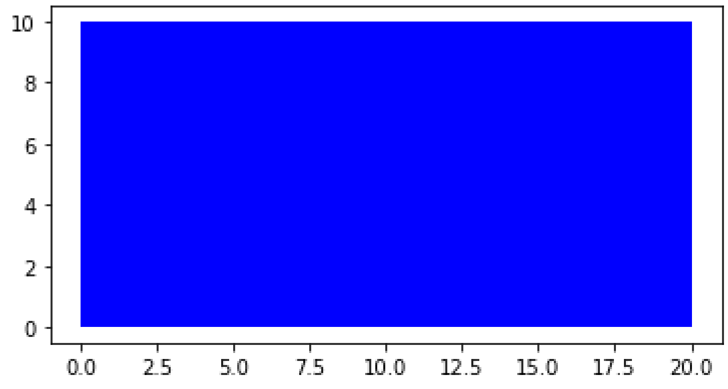
```
The euler number is 2.718
The golden ratio is 1.618
The pi number is 3.14
```

Creating a Class to draw a Rectangle

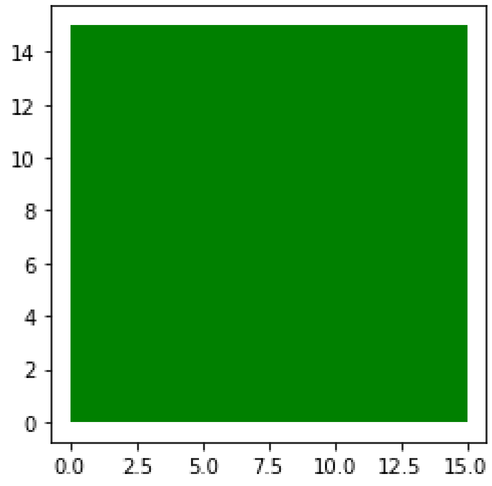
In [1]:

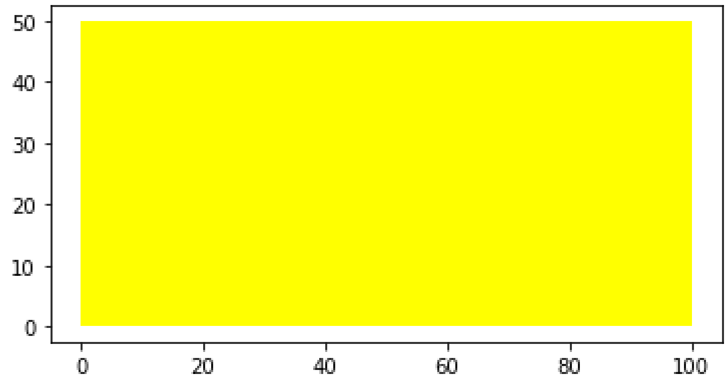
```
# Creating a class to draw a rectangle
class Rectangle(object):

    # Contructor
    def __init__(self, width, height, color):
        self.width = width
        self.height = height
        self.color = color

    # Method
    def drawRectangle(self):
        plt.gca().add_patch(plt.Rectangle((0, 0), self.width, self.height, fc=self.color))
        plt.axis('scaled')
        plt.show()

# import library to draw the Rectangle
import matplotlib.pyplot as plt
%matplotlib inline

# creating an object blue rectangle
one_Rectangle = Rectangle(20, 10, 'blue')

# Printing the object attribute width
print(one_Rectangle.width)

# Printing the object attribute height
print(one_Rectangle.height)

# Printing the object attribute color
print(one_Rectangle.color)

# Drawing the object
one_Rectangle.drawRectangle()

# Learning the methods that can be utilized on the object 'one_rectangle'
print(dir(one_Rectangle))

# We can change the properties of the rectangle
one_Rectangle.width = 15
one_Rectangle.height = 15
one_Rectangle.color = 'green'
one_Rectangle.drawRectangle()

# Using new variables, we can change the properties of the rectangle
two_Rectangle = Rectangle(100, 50, 'yellow')
two_Rectangle.drawRectangle()


```

```
20
10
```



```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribu
te__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__n
ew__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook_
_', '__weakref__', 'color', 'drawRectangle', 'height', 'width']
```





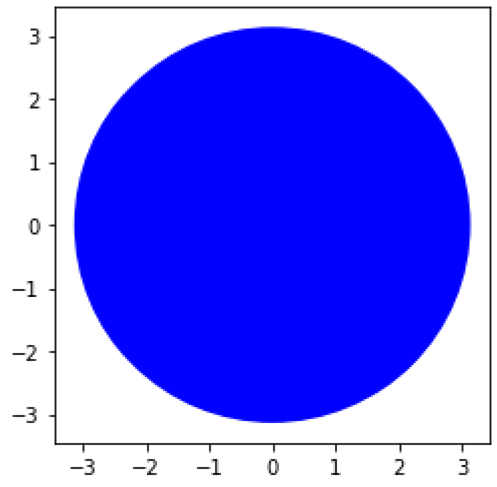
Creating a class to draw a circle

In [3]:

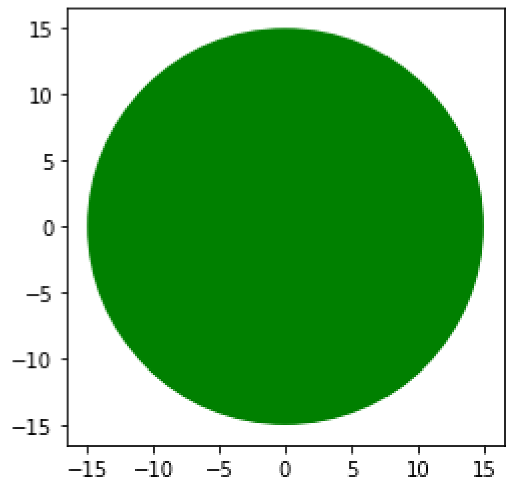
```python
# Creating a class to draw a circle
class Circle(object):

    # Contructor
    def __init__(self, radius, color):
        self.radius = radius
        self.color = color

    # Method
    def increase_radius(self, r):
        self.radius = self.radius + r
        return self.radius

    # Method
    def drawCircle(self):
        plt.gca().add_patch(plt.Circle((0, 0), self.radius, fc=self.color))
        plt.axis('scaled')
        plt.show()

# import library to draw the circle
import matplotlib.pyplot as plt
%matplotlib inline

# creating an object blue circle
one_Circle = Circle(3.14, 'blue')

# Printing the object attribute radius
print(one_Circle.radius)

# Printing the object attribute color
print(one_Circle.color)

# Drawing the object
one_Circle.drawCircle()

# Learning the methods that can be utilized on the object 'one_rectangle'
print(dir(one_Circle))

# We can change the properties of the rectangle
one_Circle.radius = 15
one_Circle.color = 'green'
one_Circle.drawCircle()

# Using new variables, we can change the properties of the rectangle
two_Circle = Circle(100, 'yellow')
print(two_Circle.radius)
print(two_Circle.color)
two_Circle.drawCircle()

# Changing the radius of the object
print('Before increment: ',one_Circle.radius)
one_Circle.drawCircle()

# Increment by 15 units
one_Circle.increase_radius(15)
print('Increase the radius by 15 units: ', one_Circle.radius)
one_Circle.drawCircle()

# Increment by 30 units
```

```
one_Circle.increase_radius(30)
print('Increase the radius by 30 units: ', one_Circle.radius)
one_Circle.drawCircle()
```

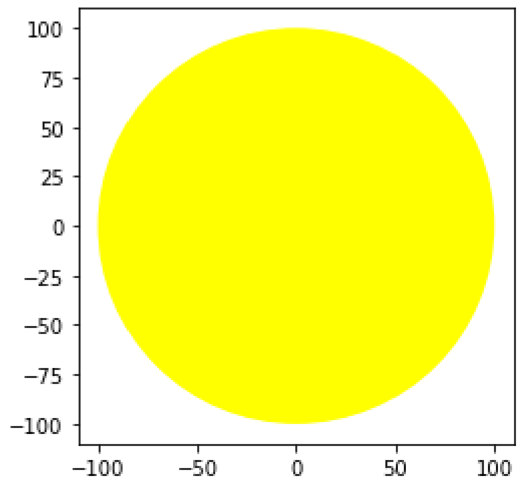
```
3.14
blue
```



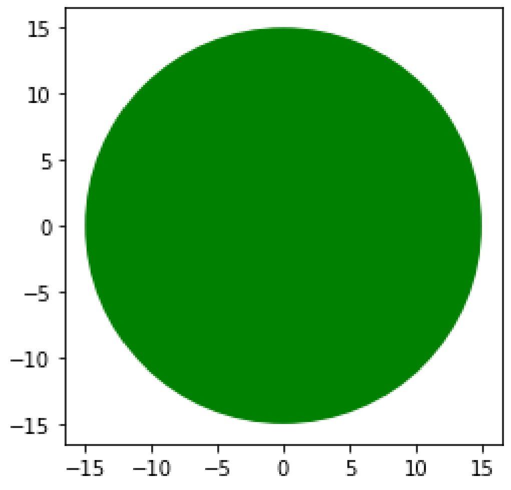
```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribu
te__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__n
ew__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook_
_', '__weakref__', 'color', 'drawCircle', 'increase_radius', 'radius']
```



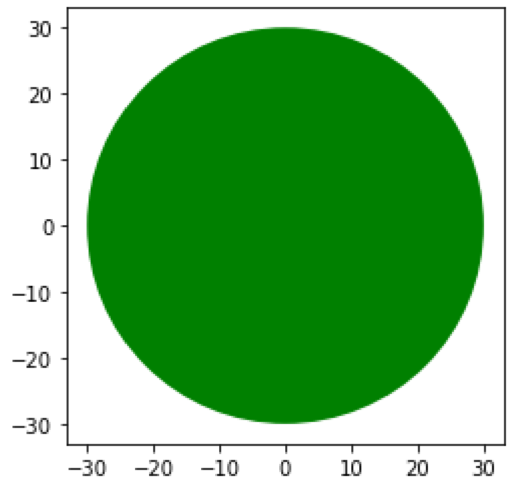
```
100
yellow
```



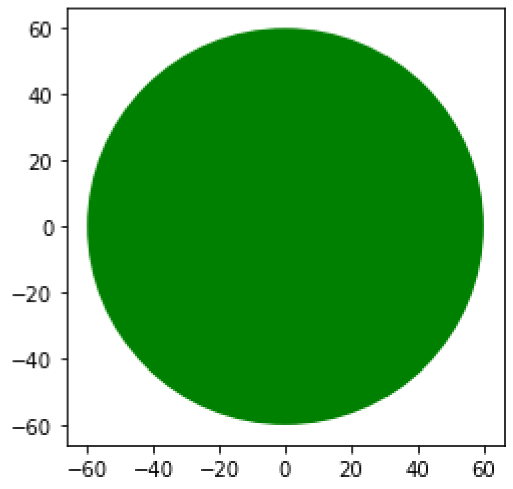
```
Before increment:  15
```



```
Increase the radius by 15 units:  30
```



```
Increase the radius by 30 units:  60
```



Some examples

In [36]:

```python
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi_number = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are special.'

special_numbers = SpecialNumbers()
print('The euler number is', getattr(special_numbers, 'euler_number'))
print('The golden ratio is', special_numbers.golden_ratio)
print('The pi number is', getattr(special_numbers, 'pi_number'))
print('The message is ', getattr(special_numbers, 'msg'))
```

```
The euler number is 2.718
The golden ratio is 1.618
The pi number is 3.14
The message is  These numbers are special.
```

In [37]:

```python
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are special.'

    def parameter(self):
        print(self.euler_constant, self.euler_number, self.pi, self.golden_ratio, self.msg)

special_numbers = SpecialNumbers()
special_numbers.parameter()
delattr(SpecialNumbers, 'msg')     # The code deleted the 'msg'.
special_numbers.parameter()        # Since the code deleted the 'msg', it returns an AttributeError.
```

```
0.577 2.718 3.14 1.618 These numbers are special.

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15364/3719874998.py in <module>
     12 special_numbers.parameter()
     13 delattr(SpecialNumbers, 'msg')     # The code deleted the 'msg'.
---> 14 special_numbers.parameter()        # Since the code deleted the 'msg', it returns an AttributeErr
or.

~\AppData\Local\Temp/ipykernel_15364/3719874998.py in parameter(self)
      7
      8     def parameter(self):
----> 9         print(self.euler_constant, self.euler_number, self.pi, self.golden_ratio, self.msg)
     10
     11 special_numbers = SpecialNumbers()

AttributeError: 'SpecialNumbers' object has no attribute 'msg'
```

In [39]:

```python
class ComplexNum:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def data(self):
        print(f'{self.a}-{self.b}j')

var = ComplexNum(3.14, 1.618)
var.data()
```

```
3.14-1.618j
```

Create a Data Classs

In [54]:

```python
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

#Use the Data class to create an object, and then execute the microorganism method
value = Data('Aspergillus', 'niger')
value.microorganism()
```

```
The name of a microorganism is in the form of Aspergillus niger.
```

Create a Child Class in Data Class

In [56]:

```python
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
    pass

value = Recombinant('Aspergillus', 'sojae')
value.microorganism()
```

```
The name of a microorganism is in the form of Aspergillus sojae.
```

Addition of init() Functions

In [4]:

```python
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
    def __init__(self, genus, species):
      Data.__init__(self, genus, species)

value = Recombinant('Aspergillus', 'sojae')
value.microorganism()
```

```
The name of a microorganism is in the form of Aspergillus sojae.
```

Addition of super() Function

In [68]:

```python
class SpecialNumbers(object):
    def __init__(self, special_numbers):
        print('6 and 28 are', special_numbers)

class PerfectNumbers(SpecialNumbers):
    def __init__(self):

        # call superclass
        super().__init__('perfect numbers.')
        print('These numbers are very special in mathematik.')

nums = PerfectNumbers()
```

```
6 and 28 are perfect numbers.
These numbers are very special in mathematik.
```

In [71]:

```python
class Animal(object):
    def __init__(self, AnimalName):
        print(AnimalName, 'lives in a farm.')

class Cow(Animal):
    def __init__(self):
        print('Cow gives us milk.')
        super().__init__('Cow')

result = Cow()
```

```
Cow gives us milk.
Cow lives in a farm.
```

In [60]:

```python
class Data:
  def __init__(self, genus, species):
    self.genus = genus
    self.species = species

  def microorganism(self):
    print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
  def __init__(self, genus, species):
   super().__init__(genus, species)        # 'self' statement in this line was deleted as different from the above codes

value = Recombinant('Aspergillus', 'sojae')
value.microorganism()
```

```
The name of a microorganism is in the form of Aspergillus sojae.
```

Addition of Properties under the super() Function

In [65]:

```python
class Data:
  def __init__(self, genus, species):
    self.genus = genus
    self.species = species

  def microorganism(self):
    print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
  def __init__(self, genus, species):
   super().__init__(genus, species)
   self.activity = 2500        # This information was adedd as a Property

value = Recombinant('Aspergillus', 'sojae')
print(f'The enzyme activity increased to {value.activity} U/mL.')
```

```
The enzyme activity increased to 2500 U/mL.
```

In [66]:

```
class Data:
  def __init__(self, genus, species):
    self.genus = genus
    self.species = species

  def microorganism(self):
    print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
  def __init__(self, genus, species, activity):
   super().__init__(genus, species)
   self.activity = activity        # This information was adedd as a Property

value = Recombinant('Aspergillus', 'sojae', 2500)
print(f'The enzyme activity increased to {value.activity} U/mL.')
```

```
The enzyme activity increased to 2500 U/mL.
```

Addition of Methods under the Child Class

In [67]:

```
class Data:
  def __init__(self, genus, species):
    self.genus = genus
    self.species = species

  def microorganism(self):
    print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
  def __init__(self, genus, species, activity):
   super().__init__(genus, species)
   self.activity = activity        # This information was adedd as a Property

  def increment(self):
    print(f'With this new recombinant {self.genus} {self.species} strain, the enzyme activity increased 2-times with {se

value = Recombinant('Aspergillus', 'sojae', 2500)
value.increment()
```

```
With this new recombinant Aspergillus sojae strain, the enzyme activity increased 2-times with 2500 U/
mL.
```