class Student:
    def __init__(self, name, exam_grade, height_in_cm):
        self.name = name
        self.grade = exam_grade
        self.height = height_in_cm
class Student:
    def __init__(self, name, exam_grade, height_in_cm):
        self.name = name
        self.grade = exam_grade
        self.height = height_in_cm

    def __str__(self):
        return "(" + self.name + ", " + str(self.grade) + ", " + str(self.height) + ")"
# create a student

```python
a = Student("Alice", 92, 160)
print(a)
```

```
(Alice, 92, 160)
```
# create a student
a = Student("Alice", 92, 160)

# access a student's information (don't do this)
print(a.height)
print(a.grade)
class Student:
    def __init__(self, name, exam_grade, height_in_cm):
        self.name = name
        self.grade = exam_grade
        self.height = height_in_cm

    def __str__(self):
        return "(" + self.name + ", " + str(self.grade) + " + " + str(self.height) + ")"

    def getName(self):
        return self.name

    def getGrade(self):
        return self.grade

    def getHeight(self):
        return self.height
# create a student
a = Student("Alice",92,160)

# access a student's information (don't do this)
print( a.height )
print( a.grade )

# access a student's information (do this)
print( a.getHeight() )
print( a.getGrade() )
student_list = [Student("Alice", 92, 160),
Student("Bob", 42, 165),
Student("Chelsea", 76, 162)]
student_list = [Student("Alice", 92, 160), \
Student("Bob", 42, 165), \
Student("Chelsea", 76, 162)]

# print all students
student_list = [Student("Alice", 92, 160), \
    Student("Bob", 42, 165), \
    Student("Chelsea", 76, 162)]

# print all students
for i in range(0,len(student_list)):
student_list = [Student("Alice", 92, 160), \\
    Student("Bob", 42, 165), \\
    Student("Chelsea", 76, 162)]

# print all students
for i in range(0,len(student_list)):  
    print(student_list[i])  # calls __str__ of Student class
student_list = [Student("Alice", 92, 160),
    Student("Bob", 42, 165),
    Student("Chelsea", 76, 162)]

# print all students
for i in range(0, len(student_list)):
    print(student_list[i])  # calls __str__ of Student class

# print all students that are failing
student_list = [Student("Alice", 92, 160), \
    Student("Bob", 42, 165), \
    Student("Chelsea", 76, 162)]

# print all students
for i in range(0,len(student_list)):  
    print(student_list[i])  # calls __str__ of Student class

# print all students that are failing
for i in range(0,len(student_list)):  

student_list = [Student("Alice", 92, 160),  
                 Student("Bob", 42, 165),  
                 Student("Chelsea", 76, 162)]

# print all students
for i in range(0, len(student_list)):
    print(student_list[i])  # calls __str__ of Student class

# print all students that are failing
for i in range(0, len(student_list)):
    if student_list[i].getGrade() < 65:
        print(student_list[i])
class Student:
    ...

    def isFailing(self):
        ...

    ...

class Student:
    ...

    def isFailing(self):
        return self.grade < 65

    ...
    ...
student_list = [Student("Alice", 92, 160), \n    Student("Bob", 42, 165), \n    Student("Chelsea", 76, "162")]

# print all students
for i in range(0,len(student_list)):
    print student_list[i]  # calls __str__ of Student class

# print all students that are failing
for i in range(0,len(student_list)):
    if( student_list[i].getGrade() < 65 ):
        print student_list[i]

# print all students that are failing (better)
for i in range(0,len(student_list)):
    if( student_list[i].isFailing() ):
        print( student_list[i] )
student_list = [Student("Alice", 92, 160),  
    Student("Bob", 42, 165),  
    Student("Chelsea", 76, "162")]

# print all exam scores in sorted order
student_list = [Student("Alice", 92, 160), \
    Student("Bob", 42, 165), \
    Student("Chelsea", 76, "162")]

# print all exam scores in sorted order
student_list.sort()
student_list = [Student("Alice", 92, 160), \n    Student("Bob", 42, 165), \n    Student("Chelsea", 76, "162")]

# print all exam scores in sorted order
student_list.sort()
student_list = [Student("Alice", 92, 160),
    Student("Bob", 42, 165),
    Student("Chelsea", 76, "162")]

# print all exam scores in sorted order
def key_grade(student):  # not a function of class Student!
    return student.getGrade()}
student_list = [Student("Alice", 92, 160), \
    Student("Bob", 42, 165), \
    Student("Chelsea", 76, "162")]

# print all exam scores in sorted order
def key_grade(student): # not a function of class Student!
    return student.getGrade()

student_list.sort(key=key_grade)
student_list = [Student("Alice", 92, 160), \
    Student("Bob", 42, 165), \
    Student("Chelsea", 76, "162")]

# print all exam scores in sorted order
def key_grade(student):  # not a function of class Student!
    return student.getGrade()

student_list.sort(key=key_grade)

for i in range(0,len(student_list)):
    print(student_list[i])

(Bob, 42, 165)
(Chelsea, 76, 162)
(Alice, 92, 160)
# --- DRILL ---
# Build a “Course” class
#
# The constructor should take as input a number N and create a list of N “Students”. Each student can have a random string for a name, a random grade [0,100] and a random height [150,200]
#
# The member function “__str__” should print all of the students in the class by calling “Students” __str__ function
# --- DRILL ---
# Build a “Course” class
#
# The constructor should take as input a number N and create a list of N “Students”.
# Each student can have a random string for a name, a random grade [0,100] and a random height [150,200]
#
# The member function “__str__” should print all of the students in the class by calling “Students” __str__ function
class Tree:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def draw(self):
        # draw tree trunk
        D.append(draw.Rectangle(self.x-4, self.y-35, 8, 30, fill='brown'))

        # draw tree top
        D.append(draw.Circle(self.x, self.y, 12, fill='green'))
class Tree:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def draw(self):
        # draw tree trunk
        D.append(draw.Rectangle(self.x-4, self.y-35, 8, 30, fill='brown'))

        # draw tree top
        D.append(draw.Circle(self.x, self.y, 12, fill='green'))

D = draw.Drawing(400, 400, origin='center')
tree = Tree(0,0)
tree.draw()
D
ordering
# make and draw a forest

def create_forest(num_trees):
    forest = []
    for i in range(num_trees):
        x = randint(-200, 200)
        y = randint(-50, 50)
        t = Tree(x, y)
        forest.append(t)
    forest.sort(key=key_y, reverse=True)
    return forest

def key_y(tree):
    return tree.getY()

def draw_forest(forest):
    D = draw.Drawing(400, 400, origin='center') # define drawing canvas
    forest = create_forest(30)
    draw_forest(forest)
    D
# make and draw a forest

def create_forest(num_trees):
    forest = []
    for i in range(num_trees):
        x = randint(-200, 200)
        y = randint(-50, 50)
        t = Tree(x, y)
        forest.append(t)
    forest.sort(key=key_y, reverse=True)
    return forest

def key_y(tree):
    return tree.getY()

def draw_forest(forest):
    D.append(draw.Rectangle(-200, -200, 400, 400, fill='#ADD8E6'))  # blue
    D.append(draw.Rectangle(-200, -100, 400, 150, fill='#90EE90'))  # green
    for tree in forest:
        tree.draw()

D = draw.Drawing(400, 400, origin='center')  # define drawing canvas
forest = create_forest(30)
draw_forest(forest)
D
# make and draw a forest

def create_forest(num_trees):
    forest = []
    for i in range(num_trees):
        x = randint(-200, 200)
        y = randint(-50, 50)
        t = Tree(x, y)
        forest.append(t)
    forest.sort(key=key_y, reverse=True)
    return forest

def key_y(tree):
    return tree.getY()

def draw_forest(forest):
    D = draw.Drawing(400, 400, origin='center')  # define drawing canvas
    forest = create_forest(30)
    draw_forest(forest)
    D
# make and draw a forest

def create_forest(num_trees):
    forest = []
    for i in range(num_trees):
        x = randint(-200, 200)
        y = randint(-50, 50)
        t = Tree(x, y)
        forest.append(t)

    # sort by height (y) top to bottom
    forest.sort(key=key_y, reverse=True)
    return forest

def key_y(tree):
    return tree.getY()

def draw_forest(forest):
    D.append(draw.Rectangle(-200, -200, 400, 400, fill='#ADD8E6'))  # blue
    D.append(draw.Rectangle(-200, -100, 400, 150, fill='#90EE90'))  # green
    for tree in forest:
        tree.draw()  # trees

D = draw.Drawing(400, 400, origin='center')  # define drawing canvas
forest = create_forest(30)
draw_forest(forest)
D
# make and draw a forest

def create_forest(num_trees):
    forest = []
    for i in range(num_trees):
        x = randint(-200, 200)
        y = randint(-50, 50)
        t = Tree(x, y)
        forest.append(t)

    forest.sort(key=key_y, reverse=True) # sort by height (y) top to bottom

    return forest

def key_y(tree):
    return tree.getY()

def draw_forest(forest):

D = draw.Drawing(400, 400, origin='center') # define drawing canvas
forest = create_forest(30)
draw_forest(forest)
D
# make and draw a forest

def create_forest(num_trees):
    forest = []
    for i in range(num_trees):
        x = randint(-200, 200)
        y = randint(-50, 50)
        t = Tree(x, y)
        forest.append(t)

    forest.sort(key=key_y, reverse=True)  # sort by height (y) top to bottom
    return forest

def key_y(tree):
    return tree.getY()

def draw_forest(forest):

D = draw.Drawing(400, 400, origin='center')  # define drawing canvas
forest = create_forest(30)
draw_forest(forest)
D
# make and draw a forest

def create_forest(num_trees):
    forest = []
    for i in range(num_trees):
        x = randint(-200, 200)
        y = randint(-50, 50)
        t = Tree(x, y)
        forest.append(t)
    forest.sort(key=key_y, reverse=True)  # sort by height (y) top to bottom
    return forest

def key_y(tree):
    return tree.getY()

def draw_forest(forest):
    D.append(draw.Rectangle(-200, -200, 400, 400, fill='#ADD8E6'))  # blue
    D.append(draw.Rectangle(-200, -100, 400, 150, fill='#90EE90'))  # green
    for tree in forest:
        tree.draw()

D = draw.Drawing(400, 400, origin='center')  # define drawing canvas
forest = create_forest(30)
draw_forest(forest)
D
# make and draw a forest

def create_forest(num_trees):
    forest = []
    for i in range(num_trees):
        x = randint(-200, 200)
        y = randint(-50, 50)
        t = Tree(x, y)
        forest.append(t)

    forest.sort(key=key_y, reverse=True) # sort by height (y) top to bottom
    return forest

def key_y(tree):
    return tree.getY()

def draw_forest(forest):
    D.append(draw.Rectangle(-200, -200, 400, 400, fill='#ADD8E6')) # blue
    D.append(draw.Rectangle(-200, -100, 400, 150, fill='#90EE90')) # green

D = draw.Drawing(400, 400, origin='center') # define drawing canvas
forest = create_forest(30)
draw_forest(forest)
D
def create_forest(num_trees):
    forest = []
    for i in range(num_trees):
        x = randint(-200, 200)
        y = randint(-50, 50)
        t = Tree(x, y)
        forest.append(t)
    forest.sort(key=key_y, reverse=True)  # sort by height (y) top to bottom
    return forest

def key_y(tree):
    return tree.getY()

def draw_forest(forest):
    D.append(draw.Rectangle(-200, -200, 400, 400, fill='#ADD8E6'))  # blue
    D.append(draw.Rectangle(-200, -100, 400, 150, fill='#90EE90'))  # green
    for tree in forest:
        tree.draw()  # trees

D = draw.Drawing(400, 400, origin='center')  # define drawing canvas
forest = create_forest(30)
draw_forest(forest)
D