Recitation 05
-
Functions

Exercise 1 - (Gaddis 5.5) Property Tax

A city collects property taxes on the assessment value of property, which is 60 percent of the property's actual value. For example, if an acre of land is valued at $10,000, its assessment value is $6,000. The property tax is then 72ï¿½ for each $100 of the assessment value. The tax for the acre assessed at $6,000 will be $43.20. Write a program that asks for the actual value of a piece of property and calls a function `computePropertyTax` to calculate and display the assessment value and property tax.

Exercise 2 - (Gaddis 5.8) Paint Job Estimator

A painting company has determined that for every 112 square feet of wall space, one gallon of paint and eight hours of labor will be required. The company charges $35.00 per hour for labor. Write a program that asks the user to enter the square feet of wall space to be painted and the price of the paint per gallon. The program should call the following functions to calculate and display results:

- `calculate_paint_volume` computes the number of gallons of paint required
- `calculate_labor_hours` computes the hours of labor required
- `calculate_paint_cost` computes the cost of the paint
- `calculate_labor_cost` computes the labor charges
- `total_cost` computes the total cost of the paint job

Exercise 3 - (Gaddis 5.12) Maximum of Two Values

Write a function named `max` that accepts two integer values as arguments and returns the value that is the greater of the two. For example, if `7` and `12` are passed as arguments to the function, the function should return `12`. Use the function in a program that prompts the user to enter two integer values. The program should display the value that is the greater of the two.

Exercise 4 - (Gaddis 5.13) Falling Distance

When an object is falling because of gravity, the following formula can be used to determine the distance the object falls in a specific time period:

$$d = \frac{1}{2} \cdot g \cdot t^2$$

The variables in the formula are as follows: d is the distance in meters, g is the gravity constant (equal to $9.81m \cdot s^{-2}$), and t is the amount of time, in seconds, that the object has been falling.

Write a function named `falling_distance` that accepts an object's falling time (in seconds) as an argument. The function should return the distance, in meters, that the object has fallen during that time interval. Write a program that calls the function in a loop that passes the values 1 through 10 as arguments and displays the return value.

Exercise 5 - (Gaddis 5.15) Test Average and Grade

Write a program that asks the user to enter five test scores. The program should display a letter grade for each score and the average test score. Write the following functions in the program:

- `calc_average` - This function should accept five test scores as arguments and return the average of the scores.
- `determine_grade` - This function should accept a test score as an argument and return a letter grade for the score based on the following grading scale:

Score	Letter Grade
90-100	A
80-89	B
70-79	C
60-69	D
Below 60	F

Exercise 6 - (Gaddis 5.17) Prime Numbers

A prime number is a number that is only evenly divisible by itself and 1. For example, the number 5 is prime because it can only be evenly divided by 1 and 5. The number 6, however, is not prime because it can be divided evenly by 1, 2, 3, and 6.

Write a Boolean function named `is_prime` which takes an integer as an argument and returns `True` if the argument is a prime number, or `False` otherwise. Use the function in a program that prompts the user to enter a number and then displays a message indicating whether the number is prime.

Exercise 7 - (Gaddis 5.18) Prime Number List

This exercise assumes that you have already written the `is_prime` function in Exercise 6. Write another program that calls the `is_prime` function in a loop to display all prime numbers between 1 to 100.

Turtle module

"Turtle Graphics" is a method of producing pictures by instructing a cursor (called a *turtle*) to move around on a Cartesian coordinate plane. The *turtle* can move left, right and forward and carries a pen around with it which can be used to draw lines to the canvas. By calling a series of functions we can cause the *turtle* to trace out shapes and figures on the screen.

In order to start working with turtle graphics we need to gain access to some graphics-specific functions that are not part of the standard Python language. We can do this by asking Python to import these functions from a function library, which is the **turtle module**:

```
import turtle
```

Link to turtle documentation

List of useful functions:

- `turtle.setup(500, 500)`
  - **setup**: sets up the screen size (width=500, height=500) - measured in pixels
  - the starting point of the *turtle* is the center of the screen (0, 0)
  - the *turtle* is facing to the right
- `turtle.forward(100)`
  - **forward**: tells the *turtle* to move forward by a desired number of units (pixels)
- `turtle.right(90)`
  - **right**: tells the *turtle* to adjust its heading to the right by a desired number of degrees
- `turtle.left(45)`
  - **left**: tells the *turtle* to adjust its heading to the left by a desired number of degrees
- `turtle.goto(100, 50)`
  - **goto**: instructs the *turtle* cursor to move to a specific position on the canvas
  - note that the *turtle*'s heading is unaffected by this function (i.e. if the *turtle* is facing right and then is told to move to a particular coordinate it will still be facing to the right when it gets to that location)
- `turtle.setheading(90)`
  - **setheading**: tells the *turtle* to turn and face a specific direction (expressed as an angle)
  - 0=right, 90=up, 180=left, 270=down

Turtle module

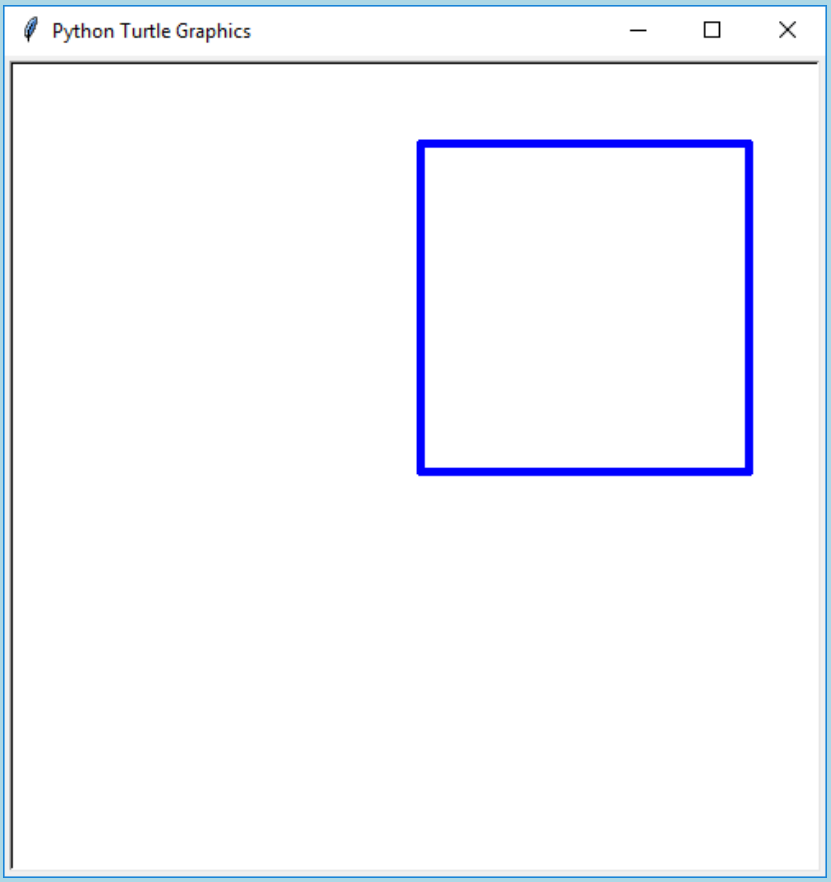
- `turtle.penup()`
  - **penup**: tells the *turtle* to pick up its pen
  - no "ink" will be drawn to the canvas if the *turtle* is asked to move from this point forward (this function takes no arguments)
- `turtle.pendown()`
  - **pendown**: tells the *turtle* to put down its pen
  - resumes drawing "ink" to the screen
- `turtle.pensize(0.1)`
  - **pensize**: sets the line thickness (default=1)
- `turtle.pencolor("red")` or `turtle.pencolor(1.0, 0, 0)`
  - **pencolor**: sets the color of the pen
  - can use a *colorstring*: List here
  - or RGB encoding: 3 float numbers (from 0.0 to 1.0) representing the intensity for Red, Green and Blue
- `turtle.fillcolor("yellow")`
  - **fillcolor**: sets the fillcolor
  - can also use RGB encoding
- `turtle.begin_fill()`
  - **begin_fill**: tells Python to start filling
- `turtle.end_fill()`
  - **end_fill**: tells Python to stop filling
- `turtle.hideturtle()`
  - **hideturtle**: makes the *turtle* invisible
- `turtle.speed(5)`
  - **speed**: sets the *turtle*'s speed to an integer value in the range 0..10
  - can also use a *speedstring*
  - `"fastest"`: 0
  - `"fast"`: 10
  - `"normal"`: 6
  - `"slow"`: 3
  - `"slowest"`: 1

Turtle module

- `turtle.tracer(0)`
  - **tracer**: turns *turtle* animation on(1) or off(0)
- `turtle.update()`
  - **update**: performs a Screen update
  - to be used when tracer is turned off

Turtle example

```
1  import turtle
2
3  #window 500x500
4  turtle.setup(500,500)
5
6  #hide cursor
7  turtle.hideturtle()
8
9  #pen color is blue
10 turtle.pencolor("blue")
11
12 #change pen size
13 turtle.pensize(5)
14
15 for i in range(4):
16     turtle.forward(200)
17     turtle.left(90)
```



Exercise 8 - Regular convex polygons

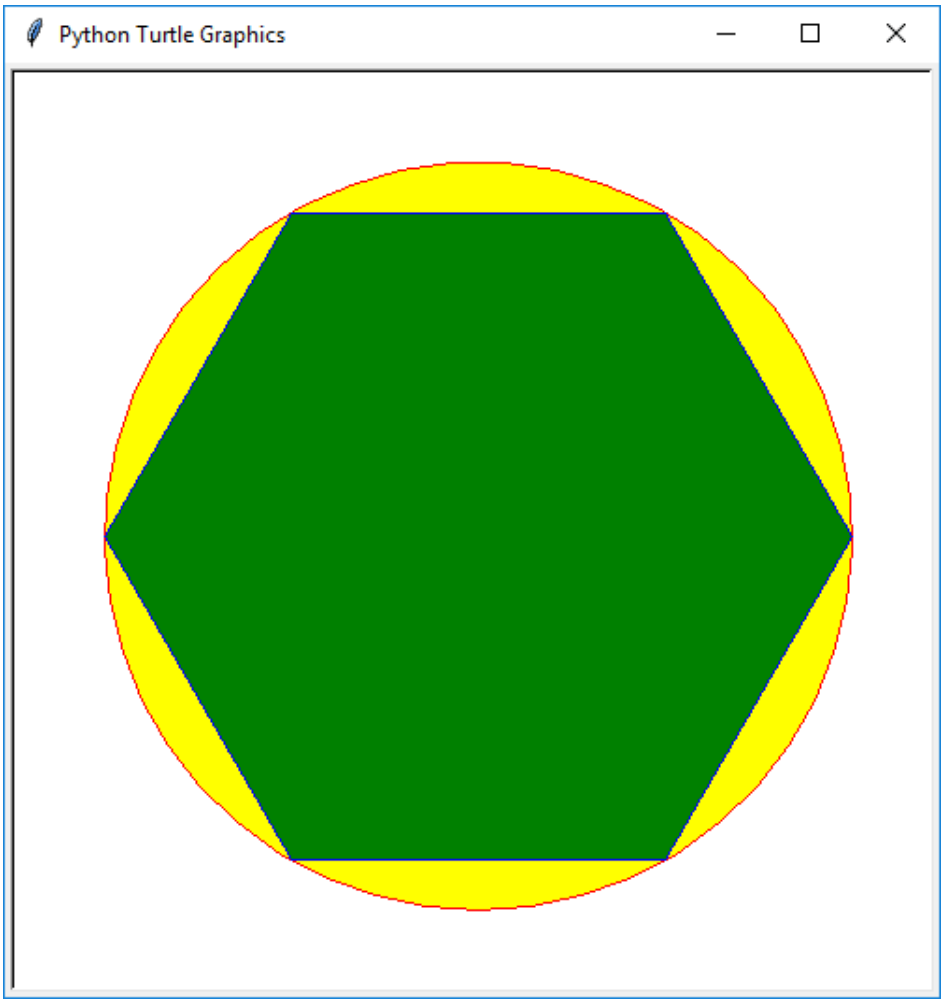
Write a function `reg_conv_polygon` that:

- takes two arguments
  - the number N of edges
  - the radius R of the circumscribed circle of the polygon
- and draws the corresponding polygon

Details about the regular convex polygon characteristics can be found **here**.

Then write a program asking that asks the user for the parameters and that uses this function.

Write a similar function `filled_reg_conv_polygon` that draws the corresponding filled polygon.



Exercise 9 - Regular star polygons

Write a function `reg_star_polygon` that:

- takes three arguments
  - the number p of edges
  - the density q
  - the radius R of the circumscribed circle of the polygon
- and draws the corresponding polygon

Details about the regular star polygon characteristics can be found **here**.

Then write a program asking that asks the user for the parameters and that uses this function.

Write a similar function `filled_reg_star_polygon` that draws the corresponding filled polygon.