Every Boilermaker Engineer Codes: 101
Entry-Level Programming in Python

Lecture 05

Dr. John H. Cole
<jhcole@purdue.edu>

Purdue University®

College of Engineering

Spring 2021

Part I

Turtle Graphics

Outline

Importing Modules

- modules contain pre-written code (like functions)
- use modules in your programs by importing them
- there are several ways to import modules
    1. `from <module_name> import *`
    2. `from <module_name> import <function_name>`
    3. `import <module_name>`
    4. and others ...
- we'll cover this more next week

Turtle Graphics

- turtle graphics is a module in the standard library
- it supplies code for a simple graphics drawing program
- read more here: docs.python.org/3/library/turtle.html
- we will import turtle graphics like this:
  `from turtle import *`

Do not name your file turtle.py!

Editor - turtle_demo.py

```python
from turtle import *
forward(100)
done()
```

setup AND bgcolor

`setup(width, height)` set the width and height of the canvas

`bgcolor(*args)` set the background color; *args can be

- named color (e.g. 'blue'; all 752 named colors are listed here: www.tcl.tk/man/tcl8.4/TkCmd/colors.htm)
- hexadecimal (e.g. '`#c29e0e`')
- rgb values (e.g 0.2, 0.8, 0.55)

Terminal

```
$ python3
>>> from turtle import *
>>> setup(480,360)
>>> bgcolor('chocolate4')
>>> bgcolor('#cfb991')
>>> bgcolor(0.0, 0.0, 0.0)
```

setup AND bgcolor

`setup(width, height)` set the width and height of the canvas

`bgcolor(*args)` set the background color; *args can be

- named color (e.g. 'blue'; all 752 named colors are listed here: www.tcl.tk/man/tcl8.4/TkCmd/colors.htm)
- hexadecimal (e.g. '#c29e0e')
- rgb values (e.g 0.2, 0.8, 0.55)

Terminal

```
$ python3
>>> from turtle import *
>>> setup(480,360)
>>> bgcolor('chocolate4')
>>> bgcolor('#cfb991')
>>> bgcolor(0.0, 0.0, 0.0)
```

setup AND bgcolor

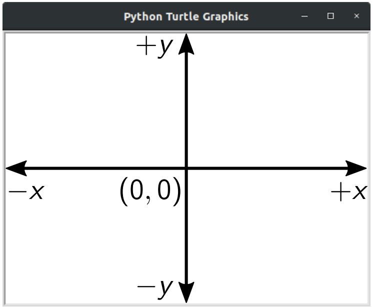
`setup(width, height)` set the width and height of the canvas

`bgcolor(*args)` set the background color; *args can be

- named color (e.g. 'blue'; all 752 named colors are listed here: www.tcl.tk/man/tcl8.4/TkCmd/colors.htm)
- hexadecimal (e.g. '#c29e0e')
- rgb values (e.g 0.2, 0.8, 0.55)

Terminal

```
$ python3
>>> from turtle import *
>>> setup(480,360)
>>> bgcolor('chocolate4')
>>> bgcolor('#cfb991')
>>> bgcolor(0.0, 0.0, 0.0)
```



setup AND bgcolor

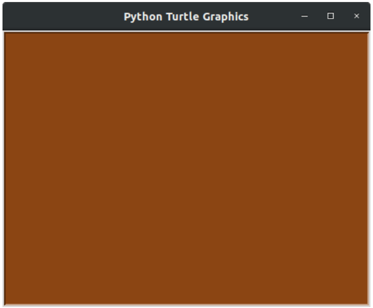
`setup(width, height)` set the width and height of the canvas

`bgcolor(*args)` set the background color; *args can be

- named color (e.g. ‘blue’; all 752 named colors are listed here: www.tcl.tk/man/tcl8.4/TkCmd/colors.htm)
- hexadecimal (e.g. ‘`#c29e0e`’)
- rgb values (e.g 0.2, 0.8, 0.55)

Terminal

```
$ python3
>>> from turtle import *
>>> setup(480,360)
>>> bgcolor('chocolate4')
>>> bgcolor('#cfb991')
>>> bgcolor(0.0, 0.0, 0.0)
```



setup AND bgcolor

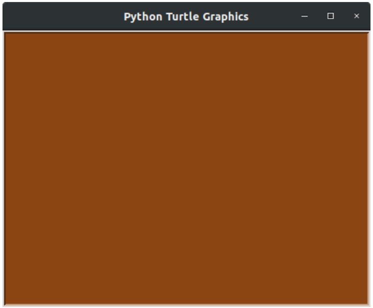
`setup(width, height)` set the width and height of the canvas

`bgcolor(*args)` set the background color; *args can be

- named color (e.g. ‘blue’; all 752 named colors are listed here: www.tcl.tk/man/tcl8.4/TkCmd/colors.htm)
- hexadecimal (e.g. ‘#c29e0e’)
- rgb values (e.g 0.2, 0.8, 0.55)

Terminal

```
$ python3
>>> from turtle import *
>>> setup(480,360)
>>> bgcolor('chocolate4')
>>> bgcolor('#cfb991')
>>> bgcolor(0.0, 0.0, 0.0)
```



setup AND bgcolor

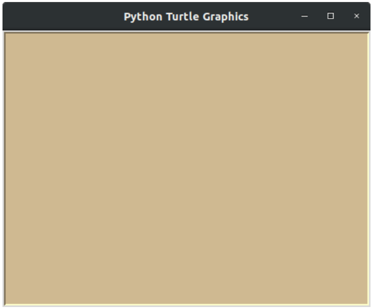
`setup(width, height)` set the width and height of the canvas

`bgcolor(*args)` set the background color; *args can be

- named color (e.g. ‘blue’; all 752 named colors are listed here: www.tcl.tk/man/tcl8.4/TkCmd/colors.htm)
- hexadecimal (e.g. ‘#c29e0e’)
- rgb values (e.g 0.2, 0.8, 0.55)

Terminal

```
$ python3
>>> from turtle import *
>>> setup(480,360)
>>> bgcolor('chocolate4')
>>> bgcolor('#cfb991')
>>> bgcolor(0.0, 0.0, 0.0)
```



setup AND bgcolor

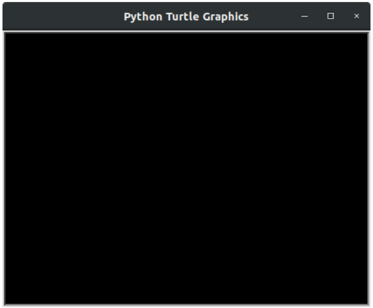
`setup(width, height)` set the width and height of the canvas

`bgcolor(*args)` set the background color; *args can be

- named color (e.g. ‘blue’; all 752 named colors are listed here: www.tcl.tk/man/tcl8.4/TkCmd/colors.htm)
- hexadecimal (e.g. ‘`#c29e0e`’)
- rgb values (e.g 0.2, 0.8, 0.55)

Terminal

```
$ python3
>>> from turtle import *
>>> setup(480,360)
>>> bgcolor('chocolate4')
>>> bgcolor('#cfb991')
>>> bgcolor(0.0, 0.0, 0.0)
```



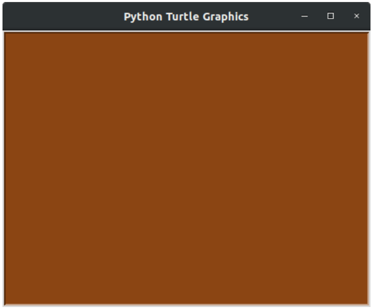
bgpic AND done

`bgpic(str)`	use an image for the background
`done()`	the last statement in a turtle graphics program

- use done at the end of your program
- do not use done in interactive mode

Terminal

```
>>> bgcolor('chocolate4')
>>> bgpic('Python-Logo.png')
>>> bgpic('maze.png')
```



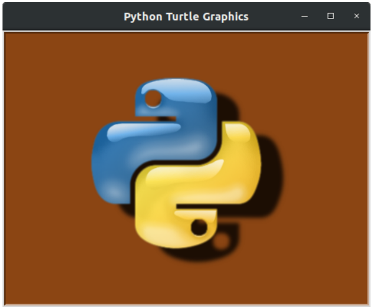
bgpic AND done

`bgpic(str)`	use an image for the background
`done()`	the last statement in a turtle graphics program

- use done at the end of your program
- do not use done in interactive mode

Terminal

```
>>> bgcolor('chocolate4')
>>> bgpic('Python-Logo.png')
>>> bgpic('maze.png')
```



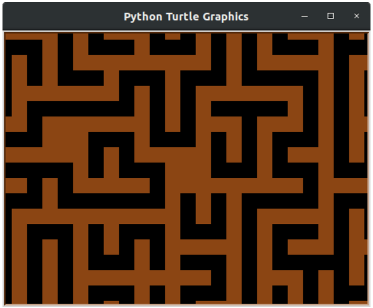
bgpic AND done

`bgpic(str)`	use an image for the background
`done()`	the last statement in a turtle graphics program

- use done at the end of your program
- do not use done in interactive mode

Terminal

```
>>> bgcolor('chocolate4')
>>> bgpic('Python-Logo.png')
>>> bgpic('maze.png')
```



Outline

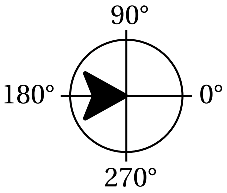
left, right, AND setheading

`left(n)`	turn counterclockwise n degrees
`right(n)`	turn clockwise n degrees
`setheading(n)`	turn to heading n

Angles increase clockwise with 0° towards right.

Terminal

```
>>> left(90) # turn north
>>> right(180) # south
>>> setheading(135) #nw
```



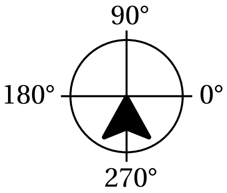
left, right, AND setheading

`left(n)`	turn counterclockwise n degrees
`right(n)`	turn clockwise n degrees
`setheading(n)`	turn to heading n

Angles increase clockwise with 0° towards right.

Terminal

```
>>> left(90) # turn north
>>> right(180) # south
>>> setheading(135) #nw
```



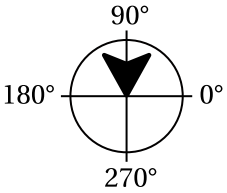
left, right, AND setheading

`left(n)`	turn counterclockwise n degrees
`right(n)`	turn clockwise n degrees
`setheading(n)`	turn to heading n

Angles increase clockwise with 0° towards right.

Terminal

```
>>> left(90) # turn north
>>> right(180) # south
>>> setheading(135) #nw
```



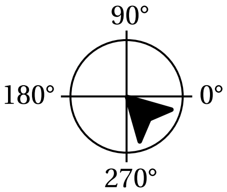
left, right, AND setheading

`left(n)`	turn counterclockwise n degrees
`right(n)`	turn clockwise n degrees
`setheading(n)`	turn to heading n

Angles increase clockwise with 0° towards right.

Terminal

```
>>> left(90) # turn north
>>> right(180) # south
>>> setheading(135) #nw
```



`forward`, `backward`, AND `goto`

`forward()`	move forward `n` pixels
`backward()`	move backward `n` pixels
`goto()`	move to `(x, y)`

Terminal

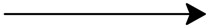
```
>>> forward(100)
>>> goto(100, 100)
>>> backward(100)
```

forward, backward, AND goto

forward()	move forward n pixels
backward()	move backward n pixels
goto()	move to (x, y)

Terminal

```
>>> forward(100)
>>> goto(100, 100)
>>> backward(100)
```

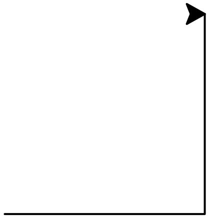


forward, backward, AND goto

`forward()`	move forward n pixels
`backward()`	move backward n pixels
`goto()`	move to `(x, y)`

Terminal

```
>>> forward(100)
>>> goto(100, 100)
>>> backward(100)
```

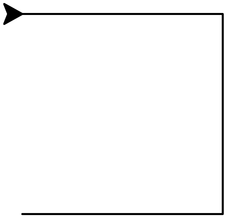


forward, backward, AND goto

forward()	move forward n pixels
backward()	move backward n pixels
goto()	move to (x, y)

Terminal

```
>>> forward(100)
>>> goto(100, 100)
>>> backward(100)
```

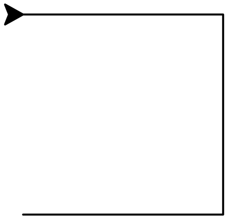


undo AND home

undo()	undo last action
home()	move to (0, 0)

Terminal

```
>>> undo()
>>> home()
```

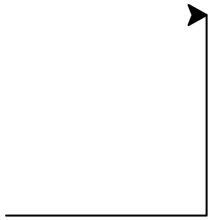


undo AND home

`undo()`	undo last action
`home()`	move to (0, 0)

Terminal

```
>>> undo()
>>> home()
```

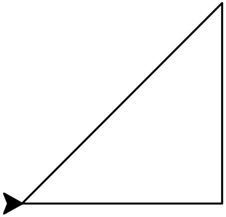


undo AND home

`undo()`	undo last action
`home()`	move to (0, 0)

Terminal

```
>>> undo()
>>> home()
```



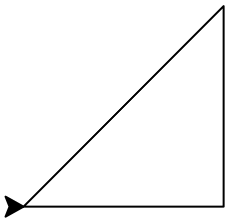
dot AND circle

`dot(width)`	draw a dot width pixels wide
`circle(radius, extent, steps)`	draw a circle

- positive radius draws ccw, negative radius draws cw
- extent sets the angular distance to draw
- steps sets the number of line segments

Terminal

```
>>> dot(10)
>>> circle(50)
>>> undo()
>>> circle(50, 180)
>>> undo()
>>> circle(50, steps=6)
```



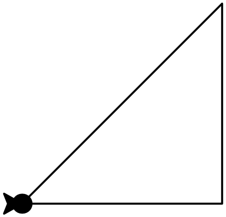
dot AND circle

`dot(width)`	draw a dot width pixels wide
`circle(radius, extent, steps)`	draw a circle

- positive radius draws ccw, negative radius draws cw
- extent sets the angular distance to draw
- steps sets the number of line segments

Terminal

```
>>> dot(10)
>>> circle(50)
>>> undo()
>>> circle(50, 180)
>>> undo()
>>> circle(50, steps=6)
```



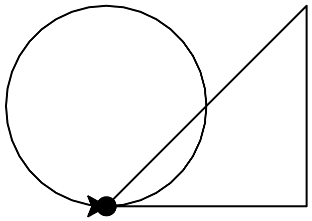
dot AND circle

`dot(width)`	draw a dot width pixels wide
`circle(radius, extent, steps)`	draw a circle

- positive radius draws ccw, negative radius draws cw
- extent sets the angular distance to draw
- steps sets the number of line segments

Terminal

```
>>> dot(10)
>>> circle(50)
>>> undo()
>>> circle(50, 180)
>>> undo()
>>> circle(50, steps=6)
```



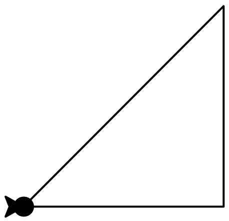
dot and circle

`dot(width)`	draw a dot width pixels wide
`circle(radius, extent, steps)`	draw a circle

- positive radius draws ccw, negative radius draws cw
- extent sets the angular distance to draw
- steps sets the number of line segments

Terminal

```
>>> dot(10)
>>> circle(50)
>>> undo()
>>> circle(50, 180)
>>> undo()
>>> circle(50, steps=6)
```



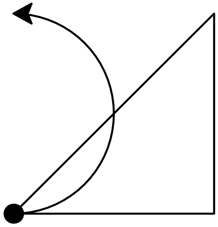
dot AND circle

`dot(width)`	draw a dot width pixels wide
`circle(radius, extent, steps)`	draw a circle

- positive radius draws ccw, negative radius draws cw
- extent sets the angular distance to draw
- steps sets the number of line segments

Terminal

```
>>> dot(10)
>>> circle(50)
>>> undo()
>>> circle(50, 180)
>>> undo()
>>> circle(50, steps=6)
```



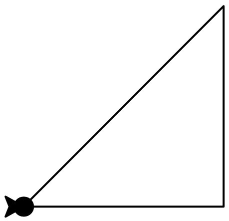
dot and circle

`dot(width)`	draw a dot width pixels wide
`circle(radius, extent, steps)`	draw a circle

- positive radius draws ccw, negative radius draws cw
- extent sets the angular distance to draw
- steps sets the number of line segments

Terminal

```
>>> dot(10)
>>> circle(50)
>>> undo()
>>> circle(50, 180)
>>> undo()
>>> circle(50, steps=6)
```



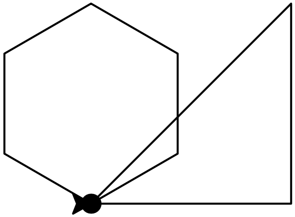
dot AND circle

`dot(width)`	draw a dot width pixels wide
`circle(radius, extent, steps)`	draw a circle

- positive radius draws ccw, negative radius draws cw
- extent sets the angular distance to draw
- steps sets the number of line segments

Terminal

```
>>> dot(10)
>>> circle(50)
>>> undo()
>>> circle(50, 180)
>>> undo()
>>> circle(50, steps=6)
```



Outline

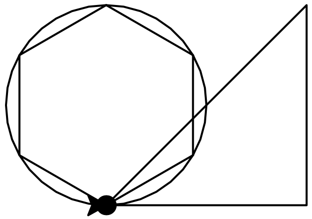
speed

speed(n) set movement speed to n (0,10)

- 1 is slowest, 10 is fast, 0 is no animation (fastest)
- any n less than 0.5 or greater than 10 sets speed to 0

Terminal

```
>>> speed(1)
>>> circle(50)
>>> undo()
>>> speed(10)
>>> circle(50)
```



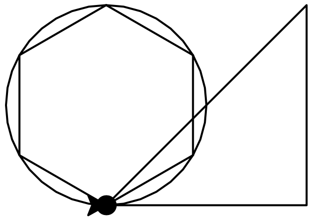
speed

`speed(n)` set movement speed to `n` (0,10)

- 1 is slowest, 10 is fast, 0 is no animation (fastest)
- any `n` less than 0.5 or greater than 10 sets speed to 0

Terminal

```
>>> speed(1)
>>> circle(50)
>>> undo()
>>> speed(10)
>>> circle(50)
```



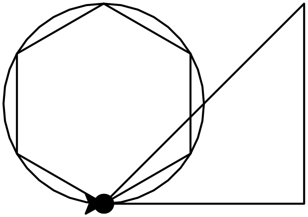
speed

speed(n) set movement speed to n (0,10)

- 1 is slowest, 10 is fast, 0 is no animation (fastest)
- any n less than 0.5 or greater than 10 sets speed to 0

Terminal

```
>>> speed(1)
>>> circle(50)
>>> undo()
>>> speed(10)
>>> circle(50)
```



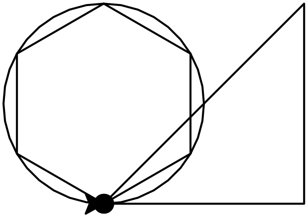
speed

`speed(n)` set movement speed to `n` (0,10)

- 1 is slowest, 10 is fast, 0 is no animation (fastest)
- any `n` less than 0.5 or greater than 10 sets speed to 0

Terminal

```
>>> speed(1)
>>> circle(50)
>>> undo()
>>> speed(10)
>>> circle(50)
```



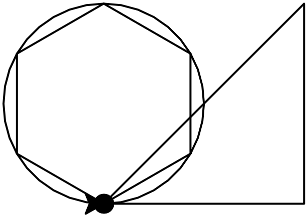
speed

`speed(n)` set movement speed to `n` (0,10)

- 1 is slowest, 10 is fast, 0 is no animation (fastest)
- any `n` less than 0.5 or greater than 10 sets speed to 0

Terminal

```
>>> speed(1)
>>> circle(50)
>>> undo()
>>> speed(10)
>>> circle(50)
```



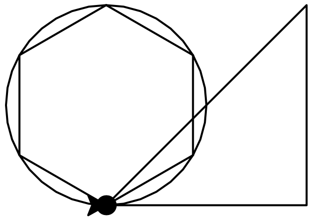
speed

`speed(n)` set movement speed to `n` (0,10)

- 1 is slowest, 10 is fast, 0 is no animation (fastest)
- any `n` less than 0.5 or greater than 10 sets speed to 0

Terminal

```
>>> speed(1)
>>> circle(50)
>>> undo()
>>> speed(10)
>>> circle(50)
```



Vec2D

`Vec2D(x,y)` create 2D vector (x,y)

For vectors a, b and scalar k:

- `a + b` vector addition
- `a - b` vector subtraction
- `a * b` inner product
- `k * a` and `a * k` multiplication with scalar
- `abs(a)` absolute value (i.e. length)
- `a.rotate(angle)` rotation

Terminal

```
>>> v1 = Vec2D(3, 0)
>>> v2 = Vec2D(0, 4)
>>> v3 = v1 + v2
>>> v3
>>> (3.00,4.00)
>>> v1 * v3
>>> 9
>>> 5 * v3
>>> (15.00, 20.00)
>>> abs(v3)
>>> 5.0
>>> v1.rotate(90)
>>> (0.00,3.00)
```

Vec2D

`Vec2D(x,y)` create 2D vector (x,y)

For vectors a, b and scalar k:

- `a + b` vector addition
- `a - b` vector subtraction
- `a * b` inner product
- `k * a` and `a * k` multiplication with scalar
- `abs(a)` absolute value (i.e. length)
- `a.rotate(angle)` rotation

Terminal

```
>>> v1 = Vec2D(3, 0)
>>> v2 = Vec2D(0, 4)
>>> v3 = v1 + v2
>>> v3
>>> (3.00,4.00)
>>> v1 * v3
>>> 9
>>> 5 * v3
>>> (15.00, 20.00)
>>> abs(v3)
>>> 5.0
>>> v1.rotate(90)
>>> (0.00,3.00)
```

Vec2D

`Vec2D(x,y)` create 2D vector (x,y)

For vectors a, b and scalar k:

- `a + b` vector addition
- `a - b` vector subtraction
- `a * b` inner product
- `k * a` and `a * k` multiplication with scalar
- `abs(a)` absolute value (i.e. length)
- `a.rotate(angle)` rotation

Terminal

```
>>> v1 = Vec2D(3, 0)
>>> v2 = Vec2D(0, 4)
>>> v3 = v1 + v2
>>> v3
>>> (3.00,4.00)
>>> v1 * v3
>>> 9
>>> 5 * v3
>>> (15.00, 20.00)
>>> abs(v3)
>>> 5.0
>>> v1.rotate(90)
>>> (0.00,3.00)
```

Vec2D

`Vec2D(x,y)` create 2D vector (x,y)

For vectors a, b and scalar k:

- `a + b` vector addition
- `a - b` vector subtraction
- `a * b` inner product
- `k * a` and `a * k` multiplication with scalar
- `abs(a)` absolute value (i.e. length)
- `a.rotate(angle)` rotation

Terminal

```
>>> v1 = Vec2D(3, 0)
>>> v2 = Vec2D(0, 4)
>>> v3 = v1 + v2
>>> v3
>>> (3.00,4.00)
>>> v1 * v3
>>> 9
>>> 5 * v3
>>> (15.00, 20.00)
>>> abs(v3)
>>> 5.0
>>> v1.rotate(90)
>>> (0.00,3.00)
```

Vec2D

`Vec2D(x,y)` create 2D vector (x,y)

For vectors a, b and scalar k:

- `a + b` vector addition
- `a - b` vector subtraction
- `a * b` inner product
- `k * a` and `a * k` multiplication with scalar
- `abs(a)` absolute value (i.e. length)
- `a.rotate(angle)` rotation

Terminal

```
>>> v1 = Vec2D(3, 0)
>>> v2 = Vec2D(0, 4)
>>> v3 = v1 + v2
>>> v3
>>> (3.00,4.00)
>>> v1 * v3
>>> 9
>>> 5 * v3
>>> (15.00, 20.00)
>>> abs(v3)
>>> 5.0
>>> v1.rotate(90)
>>> (0.00,3.00)
```

Vec2D

`Vec2D(x,y)` create 2D vector (x,y)

For vectors a, b and scalar k:

- `a + b` vector addition
- `a - b` vector subtraction
- `a * b` inner product
- `k * a` and `a * k` multiplication with scalar
- `abs(a)` absolute value (i.e. length)
- `a.rotate(angle)` rotation

Terminal

```
>>> v1 = Vec2D(3, 0)
>>> v2 = Vec2D(0, 4)
>>> v3 = v1 + v2
>>> v3
>>> (3.00,4.00)
>>> v1 * v3
>>> 9
>>> 5 * v3
>>> (15.00, 20.00)
>>> abs(v3)
>>> 5.0
>>> v1.rotate(90)
>>> (0.00,3.00)
```

Vec2D

`Vec2D(x,y)` create 2D vector (x,y)

For vectors a, b and scalar k:

- `a + b` vector addition
- `a - b` vector subtraction
- `a * b` inner product
- `k * a` and `a * k` multiplication with scalar
- `abs(a)` absolute value (i.e. length)
- `a.rotate(angle)` rotation

Terminal

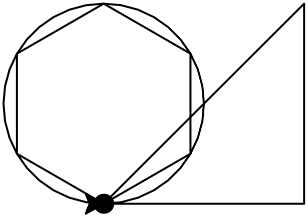
```
>>> v1 = Vec2D(3, 0)
>>> v2 = Vec2D(0, 4)
>>> v3 = v1 + v2
>>> v3
>>> (3.00,4.00)
>>> v1 * v3
>>> 9
>>> 5 * v3
>>> (15.00, 20.00)
>>> abs(v3)
>>> 5.0
>>> v1.rotate(90)
>>> (0.00,3.00)
```

position AND distance

`position()`	returns the current position as a vector
`distance(x, y)`	returns the distance between the current position and the point (x, y)

Terminal

```
>>> position()
>>> (0.00,0.00)
>>> distance(100, 100)
>>> 141.4213562373095
```

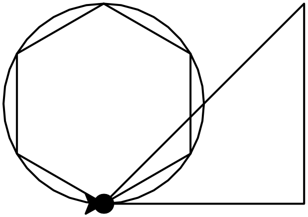


position AND distance

`position()`	returns the current position as a vector
`distance(x, y)`	returns the distance between the current position and the point (x, y)

Terminal

```
>>> position()
>>> (0.00,0.00)
>>> distance(100, 100)
>>> 141.421356237309
```



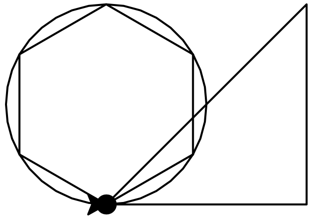
heading AND towards

`heading()`	returns the current heading angle
`towards(x, y)`	returns the angle of the line from the current position to the point (x, y)

- argument can be a Vec2D

Terminal

```
>>> heading()
>>> 0.0
>>> right(60)
>>> heading()
>>> 300.0
>>> towards(100, 100)
>>> 45.0
```

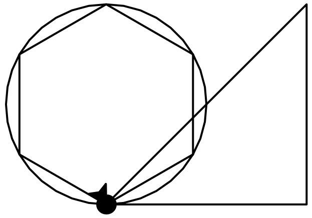


heading AND towards

`heading()`	returns the current heading angle
`towards(x, y)`	returns the angle of the line from the current position to the point (x, y) • argument can be a Vec2D

Terminal

```
>>> heading()
>>> 0.0
>>> right(60)
>>> heading()
>>> 300.0
>>> towards(100, 100)
>>> 45.0
```

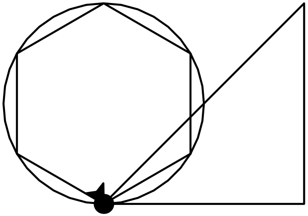


heading AND towards

`heading()`	returns the current heading angle
`towards(x, y)`	returns the angle of the line from the current position to the point (x, y) • argument can be a Vec2D

Terminal

```
>>> heading()
>>> 0.0
>>> right(60)
>>> heading()
>>> 300.0
>>> towards(100, 100)
>>> 45.0
```



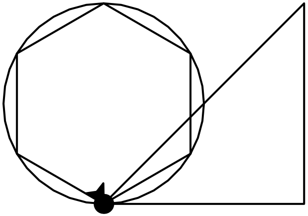
heading AND towards

`heading()`	returns the current heading angle
`towards(x, y)`	returns the angle of the line from the current position to the point (x, y)

- argument can be a Vec2D

Terminal

```
>>> heading()
>>> 0.0
>>> right(60)
>>> heading()
>>> 300.0
>>> towards(100, 100)
>>> 45.0
```



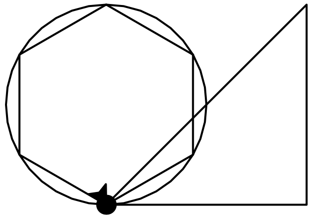
penup AND pendown

Pen is down by default.

penup()	do not draw when moved
pendown()	do draw when moved

Terminal

```
>>> home()
>>> penup()
>>> goto(0,100)
>>> pendown()
>>> goto(100,0)
```



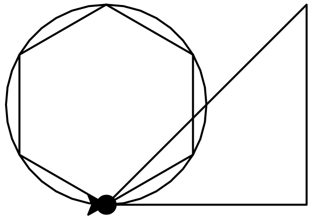
penup AND pendown

Pen is down by default.

penup()	do not draw when moved
pendown()	do draw when moved

Terminal

```
>>> home()
>>> penup()
>>> goto(0,100)
>>> pendown()
>>> goto(100,0)
```



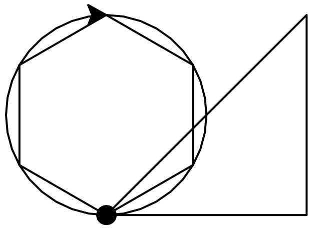
penup AND pendown

Pen is down by default.

penup()	do not draw when moved
pendown()	do draw when moved

Terminal

```
>>> home()
>>> penup()
>>> goto(0,100)
>>> pendown()
>>> goto(100,0)
```



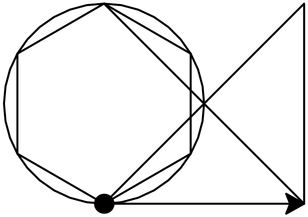
penup AND pendown

Pen is down by default.

penup()	do not draw when moved
pendown()	do draw when moved

Terminal

```
>>> home()
>>> penup()
>>> goto(0,100)
>>> pendown()
>>> goto(100,0)
```



pensize AND color

Pen is down by default.

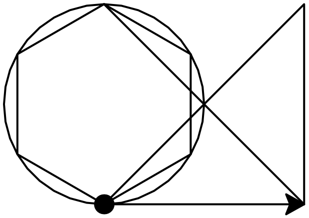
`pensize(n)` set line thickness to n

`color(color1, color2)` set the line color and fill color

- a single argument sets both line and fill color
- color1 sets the line color
- color2 sets the fill color

Terminal

```
>>> pensize(5)
>>> color('green')
>>> goto(100,100)
```



pensize AND color

Pen is down by default.

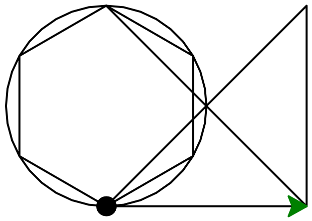
`pensize(n)` set line thickness to n

`color(color1, color2)` set the line color and fill color

- a single argument sets both line and fill color
- color1 sets the line color
- color2 sets the fill color

Terminal

```
>>> pensize(5)
>>> color('green')
>>> goto(100,100)
```



pensize AND color

Pen is down by default.

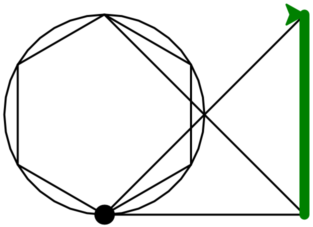
`pensize(n)` set line thickness to n

`color(color1, color2)` set the line color and fill color

- a single argument sets both line and fill color
- color1 sets the line color
- color2 sets the fill color

Terminal

```
>>> pensize(5)
>>> color('green')
>>> goto(100,100)
```

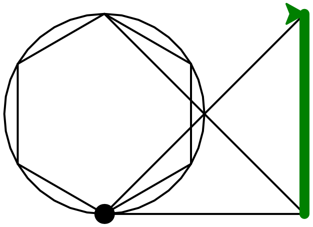


begin_fill AND end_fill

`begin_fill()`	mark the beginning of a polygon fill
`end_fill()`	mark the end, and fill the polygon

Terminal

```
>>> begin_fill()
>>> goto(50, 50)
>>> goto(100, 0)
>>> end_fill()
```

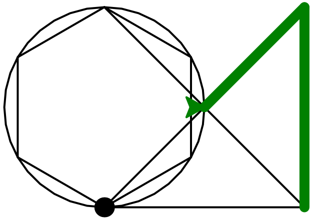


begin_fill AND end_fill

`begin_fill()`	mark the beginning of a polygon fill
`end_fill()`	mark the end, and fill the polygon

Terminal

```
>>> begin_fill()
>>> goto(50, 50)
>>> goto(100, 0)
>>> end_fill()
```

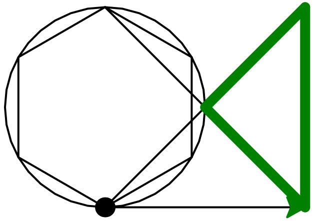


begin_fill AND end_fill

`begin_fill()`	mark the beginning of a polygon fill
`end_fill()`	mark the end, and fill the polygon

Terminal

```
>>> begin_fill()
>>> goto(50, 50)
>>> goto(100, 0)
>>> end_fill()
```

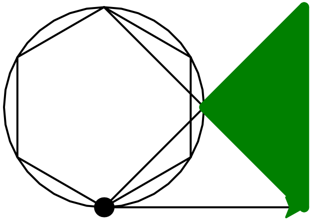


begin_fill AND end_fill

begin_fill()	mark the beginning of a polygon fill
end_fill()	mark the end, and fill the polygon

Terminal

```
>>> begin_fill()
>>> goto(50, 50)
>>> goto(100, 0)
>>> end_fill()
```

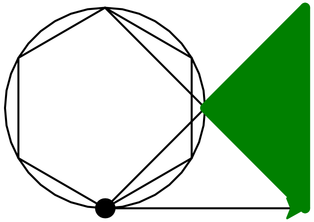


clear AND reset

`clear`	delete the drawings but leave turtle alone
`reset`	delete the drawings and reset turtle to defaults

Terminal

```
>>> setheading(135)
>>> clear()
>>> reset()
```

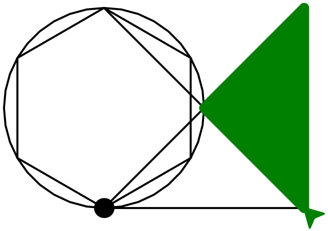


clear AND reset

clear	delete the drawings but leave turtle alone
reset	delete the drawings and reset turtle to defaults

Terminal

```
>>> setheading(135)
>>> clear()
>>> reset()
```



clear AND reset

`clear`	delete the drawings but leave turtle alone
`reset`	delete the drawings and reset turtle to defaults

Terminal

```
>>> setheading(135)
>>> clear()
>>> reset()
```

clear AND reset

`clear`	delete the drawings but leave turtle alone
`reset`	delete the drawings and reset turtle to defaults

Terminal

```
>>> setheading(135)
>>> clear()
>>> reset()
```

hideturtle AND showturtle

`hideturtle()`	make the turtle invisible
`showturtle()`	make the turtle visible (default)

Terminal

```
>>> hideturtle()
>>> goto(100, 100)
>>> showturtle()
```

hideturtle AND showturtle

`hideturtle()`	make the turtle invisible
`showturtle()`	make the turtle visible (default)

Terminal

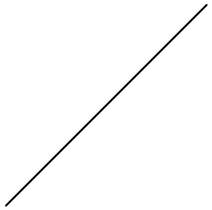
```
>>> hideturtle()
>>> goto(100, 100)
>>> showturtle()
```

hideturtle AND showturtle

`hideturtle()`	make the turtle invisible
`showturtle()`	make the turtle visible (default)

Terminal

```
>>> hideturtle()
>>> goto(100, 100)
>>> showturtle()
```

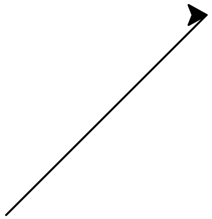


hideturtle AND showturtle

`hideturtle()`	make the turtle invisible
`showturtle()`	make the turtle visible (default)

Terminal

```
>>> hideturtle()
>>> goto(100, 100)
>>> showturtle()
```



shape AND shapesize

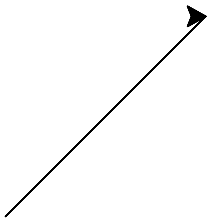
`shape(str)` ‘arrow’, ‘turtle’, ‘circle’, ‘square’, ‘triangle’, or ‘classic’

`shapesize(stretch_wid, stretch_len, outline)`

- `stretch_wid`: multiplies the shapes width by `stretch_wid`
- `stretch_len`: multiplies the shapes length by `stretch_len`
- `outline`: sets the width of the shapes outline

Terminal

```
>>> shape('turtle')
>>> shapesize(3, 1)
>>> shapesize(3, 3)
>>> color('black','green')
>>> shapesize(3, 3, 3)
```



shape AND shapesize

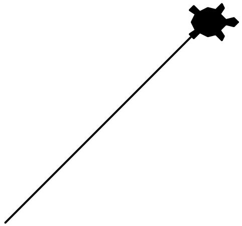
`shape(str)` ‘arrow’, ‘turtle’, ‘circle’, ‘square’, ‘triangle’, or ‘classic’

`shapesize(stretch_wid, stretch_len, outline)`

- `stretch_wid`: multiplies the shapes width by `stretch_wid`
- `stretch_len`: multiplies the shapes length by `stretch_len`
- `outline`: sets the width of the shapes outline

Terminal

```
>>> shape('turtle')
>>> shapesize(3, 1)
>>> shapesize(3, 3)
>>> color('black','green')
>>> shapesize(3, 3, 3)
```



shape AND shapesize

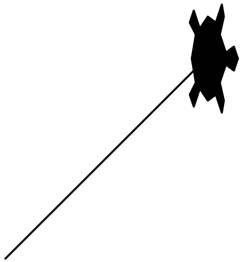
`shape(str)` 'arrow', 'turtle', 'circle', 'square', 'triangle', or 'classic'

`shapesize(stretch_wid, stretch_len, outline)`

- `stretch_wid`: multiplies the shapes width by `stretch_wid`
- `stretch_len`: multiplies the shapes length by `stretch_len`
- `outline`: sets the width of the shapes outline

Terminal

```
>>> shape('turtle')
>>> shapesize(3, 1)
>>> shapesize(3, 3)
>>> color('black','green')
>>> shapesize(3, 3, 3)
```



shape AND shapesize

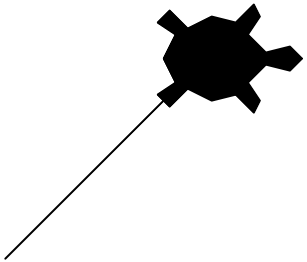
`shape(str)` 'arrow', 'turtle', 'circle', 'square', 'triangle', or 'classic'

`shapesize(stretch_wid, stretch_len, outline)`

- `stretch_wid`: multiplies the shapes width by `stretch_wid`
- `stretch_len`: multiplies the shapes length by `stretch_len`
- `outline`: sets the width of the shapes outline

Terminal

```
>>> shape('turtle')
>>> shapesize(3, 1)
>>> shapesize(3, 3)
>>> color('black','green')
>>> shapesize(3, 3, 3)
```



shape AND shapesize

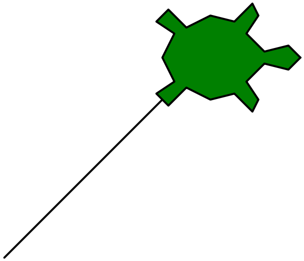
shape(str) ‘arrow’, ‘turtle’, ‘circle’, ‘square’, ‘triangle’, or ‘classic’

shapesize(stretch_wid, stretch_len, outline)

- stretch_wid: multiplies the shapes width by stretch_wid
- stretch_len: multiplies the shapes length by stretch_len
- outline: sets the width of the shapes outline

Terminal

```
>>> shape('turtle')
>>> shapesize(3, 1)
>>> shapesize(3, 3)
>>> color('black','green')
>>> shapesize(3, 3, 3)
```



shape AND shapesize

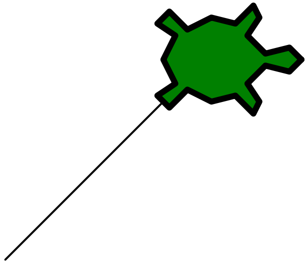
`shape(str)` ‘arrow’, ‘turtle’, ‘circle’, ‘square’, ‘triangle’, or ‘classic’

`shapesize(stretch_wid, stretch_len, outline)`

- `stretch_wid`: multiplies the shapes width by `stretch_wid`
- `stretch_len`: multiplies the shapes length by `stretch_len`
- `outline`: sets the width of the shapes outline

Terminal

```
>>> shape('turtle')
>>> shapesize(3, 1)
>>> shapesize(3, 3)
>>> color('black','green')
>>> shapesize(3, 3, 3)
```



More examples

- Turtle can do *much* more.
- To see some examples, run the turtledemo module in python

Terminal

```
$ python3 -m turtledemo
```

- More details are available at docs.python.org/3/library/turtle.html

More examples

- Turtle can do *much* more.
- To see some examples, run the turtledemo module in python

Terminal

```
$ python3 -m turtledemo
```

- More details are available at docs.python.org/3/library/turtle.html

More examples

- Turtle can do *much* more.
- To see some examples, run the turtledemo module in python

Terminal

```
$ python3 -m turtledemo
```

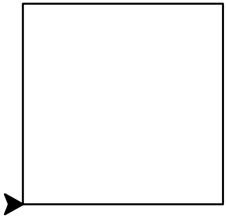
- More details are available at docs.python.org/3/library/turtle.html

Part II

Your Turn

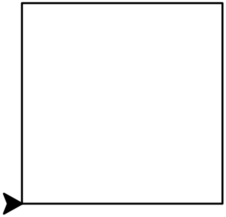
Practice Exercise 1

Draw this square. Try doing it with a loop.



Practice Exercise 1

Draw this square. Try doing it with a loop.



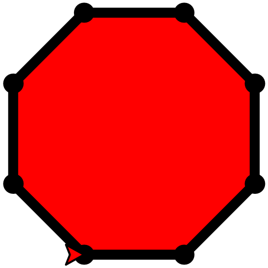
Editor - practice_1.py

```
from turtle import *
for _ in range(4):
    forward(100)
    left(90)

done()
```

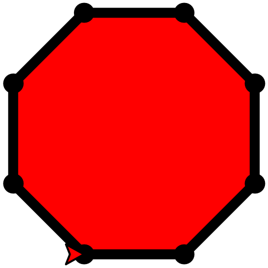
Practice Exercise 2

Draw this octagon using a loop.



Practice Exercise 2

Draw this octagon using a loop.



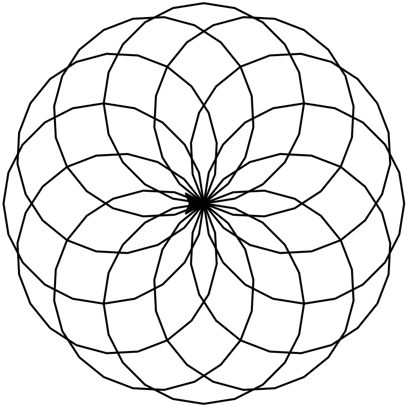
Editor - practice_2.py

```
from turtle import *
color('black', 'red')
width(5)
begin_fill()
for _ in range(8):
    forward(50)
    dot()
    left(45)
end_fill()

done()
```

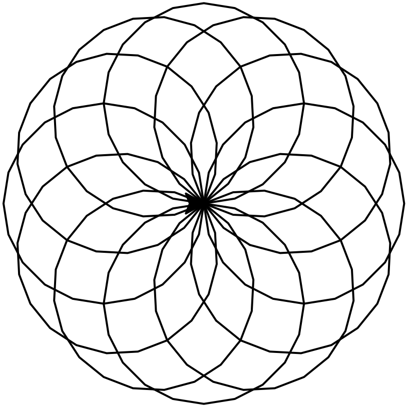
Practice Exercise 3

Draw these 12 circles.



Practice Exercise 3

Draw these 12 circles.



Editor - practice_3.py

```
from turtle import *
speed(10)
for _ in range(12):
    circle(50)
    right(30)

done()
```

Thanks for watching!