Background for lab: the ord() function

- Examples
- >>> ord('a')
- 97
- >>> ord('b')
- 98
- >>> ord('c')
- 99

- The ord function:
  - is a built-in Python function
  - returns American Standard Code for Information Interchange (ASCII)

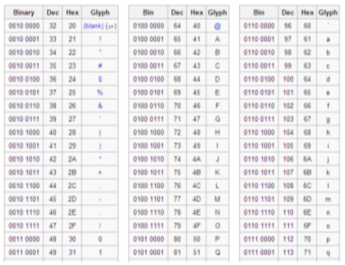
American Standard Code for Information Interchange (ASCII)

```
ASCII is a table
that tells the
computer how to
represent
characters as #s
```

ord('a') is 97

ord('2') is 50

ord('Q') is 81



Binary	Dec	Hex	Glyph	Bin	Dec	Hex	Glyph	Bin	Dec	Hex	Glyph
0010 0000	32	20	(blank) (␠)	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a ←
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q

ASCII VALUES

`abcdefghijklmnopqrstuvwxyz`

`97 99    103`

`ord('a')` is **97** **`ord('c')`** is **99**

Can you use the ord function to determine how "far" a letter is from **`'a'`**?

For example, **`'c'`** is two letters away from **`'a'`**.

How "far" is **`'g'`** from **`'a'`**?

Turtle Graphics

Python way of drawing

Python's Etch-a-Sketch

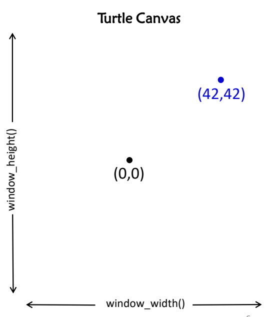
- Want graphics? In Python, we give commands to a "turtle" to draw on a digital canvas!

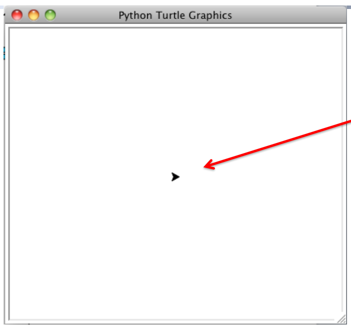
```
from turtle import *
```

https://en.wikipedia.org/wiki/Turtle_graphics

The turtle canvas

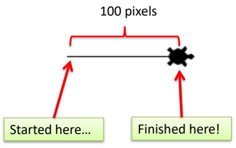
- Canvas operates in x-y coordinate plane
  - (0,0) is the center
- **`reset()`**
  - Delete any drawings, reset the screen, re-center the turtle
  - Turtle reset to face right (or east)





A real turtle?

```
reset()
shape('turtle')
forward(100)
done()
```



Example

reset()
forward(50)
- In pixels
- NOTE: **backward(n)** moves the turtl back

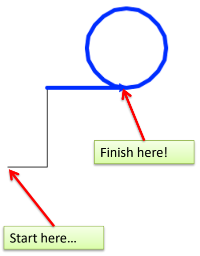
left(90)
- In degrees

forward(100)
right(90)
- In degrees

color('blue')
width(5)
- In pixels

forward(100)
circle(50)
- Starts drawing circle to the left of the turtle
- Radius is specified In pixels

done()



Pen up, Pen down

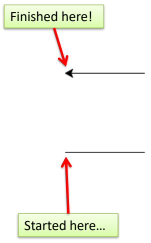
reset()
fd(100)
- Same as forward()

lt(90)
- Same as left ()

up()
- Lifts the pen off the canvas

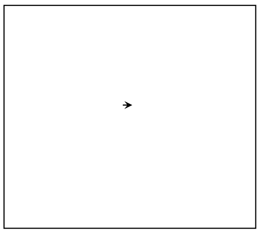
fd(100)
lt(90)
down()
- Puts the pen down on the canvas

fd(100)
done()



Exercise

(1) What does function **chai** draw?



```
def chai(size):
    """ mystery! """
    forward(size)
    left(90)
    forward(size/2.0)
    right(90)
    right(90)
    forward(size)
    left(90)
    left(90)
    forward(size/2.0)
    right(90)
    backward(size)
```

Why are there two identical commands in a row?

How could you add more to each end?

Turtle Graphics

We will be using Python's built-in turtle graphics package.

You will want to have this line in your **hw2pr3.py** file:

```
from turtle import *
```

Then you will be able to write functions using **turtle** commands:

https://docs.python.org/3/library/turtle.html

Turtle reference

degrees()	radians()	reset()
clear()	tracer(flag)	forward(distance)
backward(distance)	left(angle)	right(angle)
up()	down()	width(width)
color(*args)	begin_fill()	end_fill()
setheading(angle)	window_width()	window_height()
position()	setx(xpos)	sety(ypos)
goto(x,y)	heading()	

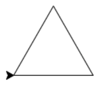
Also see link in the homework

For basic help on Python's turtle graphics module

Recursive Graphics

do...
or do not...
there is no tri ...

```
def tri():
    """ draws a polygon
    """
    forward(100)
    left(120)
    forward(100)
    left(120)
    forward(100)
    left(120)
```



(1) Could we `tri` this with recursion?

```
def tri(   ):
    """ draws a polygon """
def triRec(n=3):
  if n == 0:
    return
  else:
    forward(100)
    left(120)
    triRec(n-1)
```

(2) Could we create ***any*** regular n-gon?

Generic n-gon

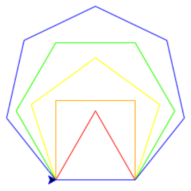
A default parameter value!

```
def ngon(nSides,curSide=0):
    """ A simple recursive function to create an arbitrary
        n-sided polygon
        Parameters:
            n - Number of sides of the polygon
            curSide - used by the recursion
    """
    if curSide >= nSides:
        return
    else:
        forward(100)
        left(360/nSides)
        ngon(nSides,curSide+1)
```

How many degrees should we turn?

ngon(nSides)

```
pencolor('red')
ngon(3)
pencolor('orange')
ngon(4)
pencolor('yellow')
ngon(5)
pencolor('green')
ngon(6)
pencolor('blue')
ngon(7)
```



Exercise

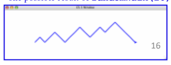
(2) Finish `randomWalk` to draw a "stock-market-type" random path of `nSteps` steps.

```
from random import *

def randomWalk(nSteps):
    """ Move for nSteps steps, randomly
        Turn 45 deg. left/up or right/down """
    if nSteps == 0:
        return
    if choice(['left','right']) == 'left':




    else:  # 'right'
```

What if we ***didn't*** turn back to face forward each time?

Ex Cr: How could you make it a bull (or a bear) market?

one possible result of **`randomWalk(20)`**



```
from turtle import *
from random import *

def randomWalk(nSteps):
    if nSteps == 0:
        return
    if choice(['left','right']) == 'left':
        left(45)
        fd(20)
        right(45)
    else:    # right
        right(45)
        fd(20)
        left(45)
    randomWalk(nSteps - 1)
```

Fast turtle!

- You can adjust the speed of the turtle
- **`tracer(n)`**
  - Sets drawing to update every "regular" nth screen update
    - Use larger values for faster updates
- **`tracer(1)`**
  - Default – Slowest update
  - To speed up drawing, set to a higher value
- **`tracer(0)`**
  - Disables screen updates.
  - After you draw, call the **`update()`** function to force drawing to appear on screen