CMPT 120
Intro to CS & Programming I

WEEK 3 (Jan. 20-24)

— *Jérémie O. Lumbroso* —

Lecture 7:
Turtles!

http://www.sfu.ca/~jlumbros/Courses/CMPT120/

One very fundamental difficulty of functions

PRINTING OR RETURNING?

Some Experimentation

```
def printSquare(x):
    print x*x
```

```
def returnSquare(x):
    return x*x
```

Enter the definitions of the above functions, then try:

```
>>> printSquare(10)
```

A 100 B 10 C None D Nothing E ERROR

```
>>> returnSquare(10)
```

A 100 B 10 C None D Nothing E ERROR

```
>>> printSquare(10) + printSquare(5)
```

A 125 B 15 C None D Nothing E ERROR

```
>>> returnSquare(10) + returnSquare(5)
```

A 125 B 15 C None D Nothing E ERROR

More Experimentation

```
def printSquare(x):
  print x*x
```

```
def returnSquare(x):
  return x*x
```

```
>>> x = printSquare(10)
```

A 100 B 10 C None D Nothing E ERROR

```
>>> y = returnSquare(10)
```

A 100 B 10 C None D Nothing E ERROR

```
>>> print x
```

A 100 B 10 C None D Nothing E ERROR

```
>>> print y
```

A 100 B 15 C None D Nothing E ERROR

```
>>> type(x)
```

A 100 B 15 C None D Nothing E ERROR

Printing vs. Returning 

- **Printing**: display something on the console
- **Returning**: transmitting a value so that it can be used for further calculations
- They **seem** similar because the Python shell in which you type stuff and get an answer always displays the value of expressions automatically
- But printing is not the same thing as returning

No Return No Type

doesn't return a value (NoneType)			returns a value

`def printSquare(x):`
`  print x*x`
	is equivalent to	`def printSquare(x):`
		`  print x*x`
		`  return`
			`def returnSquare(x):`
			`  return x*x`

- A procedure that contains **no** return statement explicitly, acts like a function that has an empty return statement (which returns no value)
- The function `type(x)` allows you to determine the type of variable `x`
- `type(printSquare(10))` is `NoneType`
- `type(returnSquare(10))` is `int` (integer)
- A procedure cannot be part of an expression
- A function (because it returns a value) can be part of an expression

Printing/Returning Strings 

- Finally, another difference even in the Python shell

```
>>> 'hello'
'hello'
>>> print 'hello'
hello
>>> type('hello')
<type 'str'>
```

- When the Python shell **displays** a value either:
  - that you typed yourself…
  - that was computed from an expression…
  - that was return by a function…

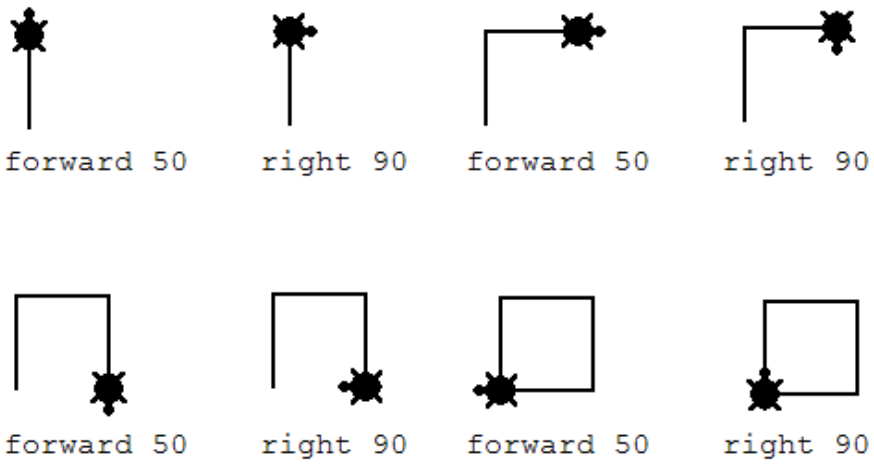
  if value is a string, it is surrounded with quotes (simple or double, doesn't matter)

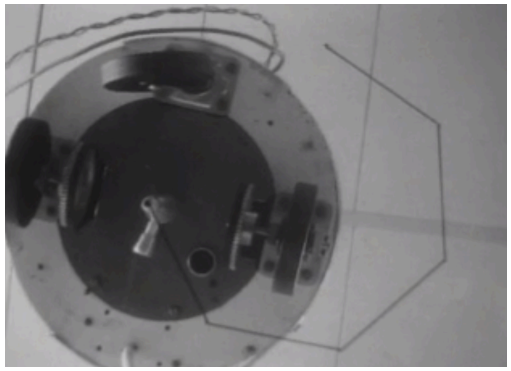
- When the Python shell **prints** a value that is a string, quotes don't appear

Understanding

Is this **important** distinction between printing and returning clear?

- **A** Yep, but I already understood it before
- **B** I was not aware of this distinction, but now I am
- **C** I am not quite sure I understand the distinction, but I'll get it
- **D** I need more examples
- **E** Why am I still in this course?? ☹

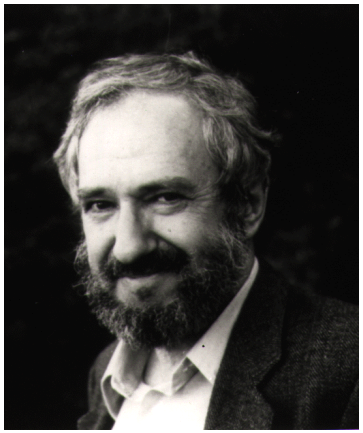




Guiding a turtle with a pen, and learning about recursive calls

TURTLES!!

LOGO



- Language created by Seymour Papert (+ others) in 60-70s
- For educational purposes ("*Third-graders should be able to use it for simple tasks with very little preparation*"), about what can be learned *by* programming, esp. mathematically
- Defining (though not essential) feature: **Turtle graphics**

IT'S TIME KIDS STARTED USING STRONG LANGUAGE.

We encourage it.

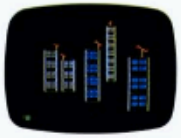
Because now the most powerful educational language is available on the Apple Personal Computer.

Presenting Apple Logo.

It's not just a programming language for computers, but a learning language for people.

Enough so that anyone, working with Apple Logo, can easily learn the programming principles once reserved for college courses.

Apple Logo encourages you to break problems into small steps, and then shows you how to make those steps automatic.



It does all this interactively.

For instance, if you accidentally type "foreword," instead of forward, Apple Logo responds with "I don't know how to foreword."

There is no such thing as a mistake with Apple Logo, only logical statements telling you what needs to be done to make the program work. So the student programs the computer. Not the computer the student.

And as you learn, Apple Logo learns with you. So whether you're a student of 5 or 55, you'll always be challenged—but not overwhelmed.

Apple Logo runs on the Apple II with 64K. And it comes from Apple, the leading personal computer company in education—with the largest library of courseware at all levels.

Apple Logo. It can make getting to know a computer the most positive of learning experiences.

Your kids will swear by it.

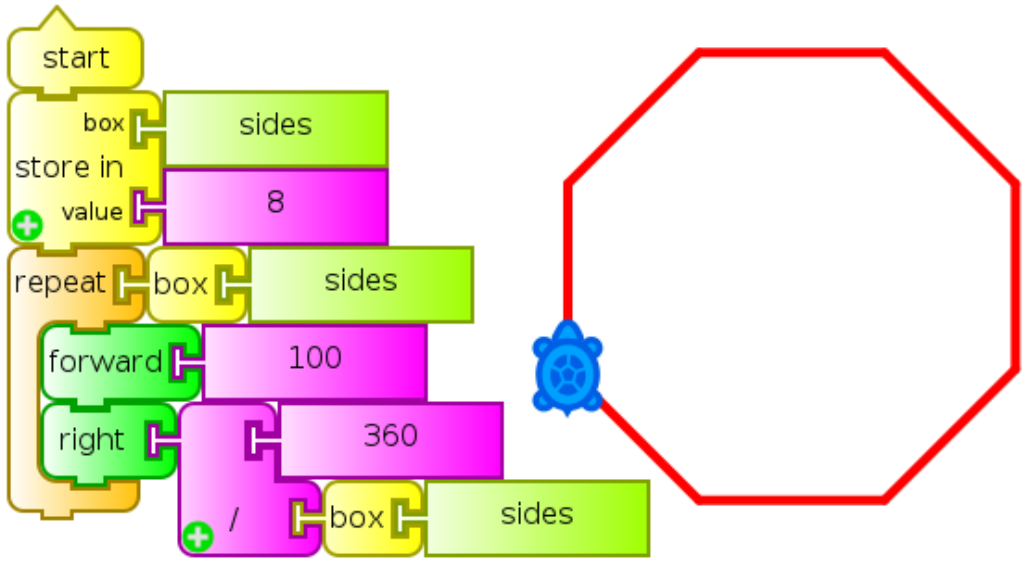
The personal computer.

apple

For more information, call (800) 538-9696. In California, call (800) 662-9238. Or write: Apple Computer Inc., 20525 Mariani Avenue, Cupertino, CA 95014.
Apple® Logo is a product of Logo Computer Systems, Inc., 222 Brunswick Boulevard, Point-Claire, Quebec, Canada H9R1A6.

Many Versions

- LOGO has been hugely popular
- Even Turtle Blocks, a block-programming version
- http://wiki.sugarlabs.org/go/Activities/Turtle_Art



`import turtle`

- We are not going to use LOGO, but Python does have a module to use a turtle
- Turtle graphics gives the ability to move a pen on the screen and make drawings; turtle commands are **procedures**, like **print** they do not return any value
- Roadmap
  - first discover the built-in constructs
  - then write our functions and loops to draw
  - make beautiful stuff

How Does Turtle Work?

- By giving commands such as "move forward", "turn left", "turn right", you are directing a little turtle on a screen
- This turtle holds a pen, and so whenever it moves, it draws a line
- It is also possible to have the pen lifted to move the turtle without drawing, or to do other actions such as filling a closed region with color

Initialize the Turtle

- Before any drawing can take place, must initialize the turtle using the following code

```
import turtle        # Let Python know you are going to use the turtle module
turtle.reset()       # Open the drawing window and place the turtle in the center
turtle.speed('fastest')   # Without this, the turtle is slow as shitake mushrooms
```

- The instruction `turtle.reset()` can be used every time you want to erase the screen and place the turtle in the center

- Be sure to change the speed as well (every time the turtle is reset)

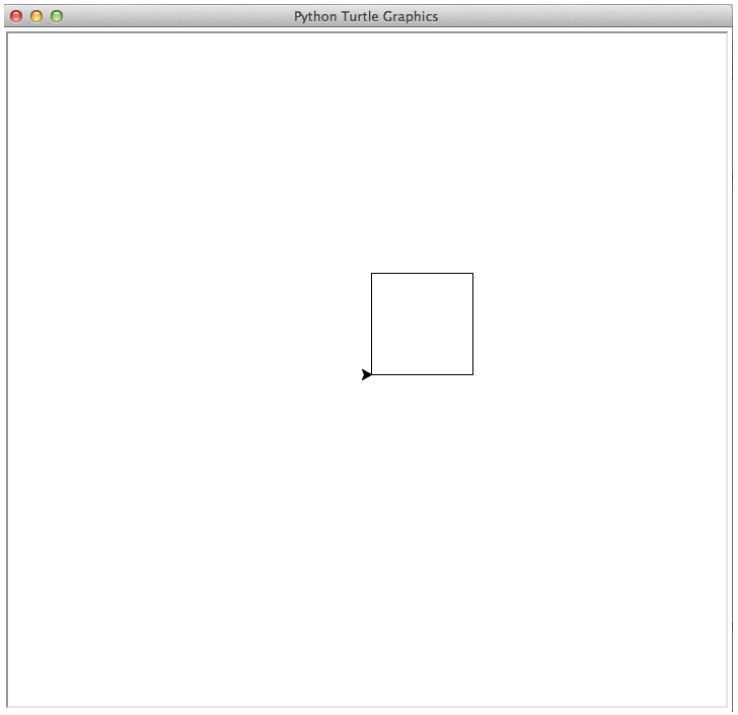
First Example

The following instructions will draw a square

```
import turtle
turtle.reset()

# A square has four sides
# range(4) = [0, 1, 2, 3]
# loop will repeat four times

for i in range(4):
  turtle.forward(100)
  turtle.left(90)
```



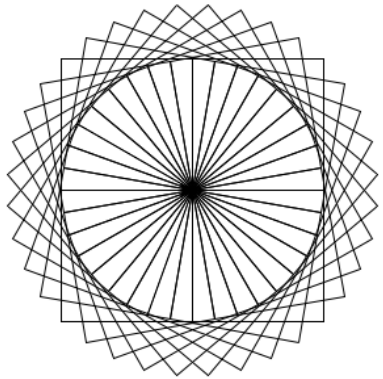
Square Function

- Because drawing a square is a useful action that we might reuse, we can write a function

```
def square():
  for i in range(4):
    turtle.forward(100)
    turtle.left(90)
```

- We can then use that function, for instance

```
for i in range(36):
  square()
  turtle.right(10)
```



Hard-coding

- "Hard-coding" means writing the value data/ parameters directly inside your code
- For the square function, we decided that the size of that square is 100; it cannot change

```
def square():
  for i in range(4):
    turtle.forward(100)
    turtle.left(90)
```

What if we want a smaller/larger square?

- "Hard-coding" is **bad** because it limits the flexibility of your code

Extreme Hard-coding

- Suppose we want to write a function `getRandomNumber` which returns a random number between 1 and 6 (a dice roll)
- Here is how not to do it:

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

http://xkcd.com/221/

Improving the Example

With the new function, we can do more extravagant figures

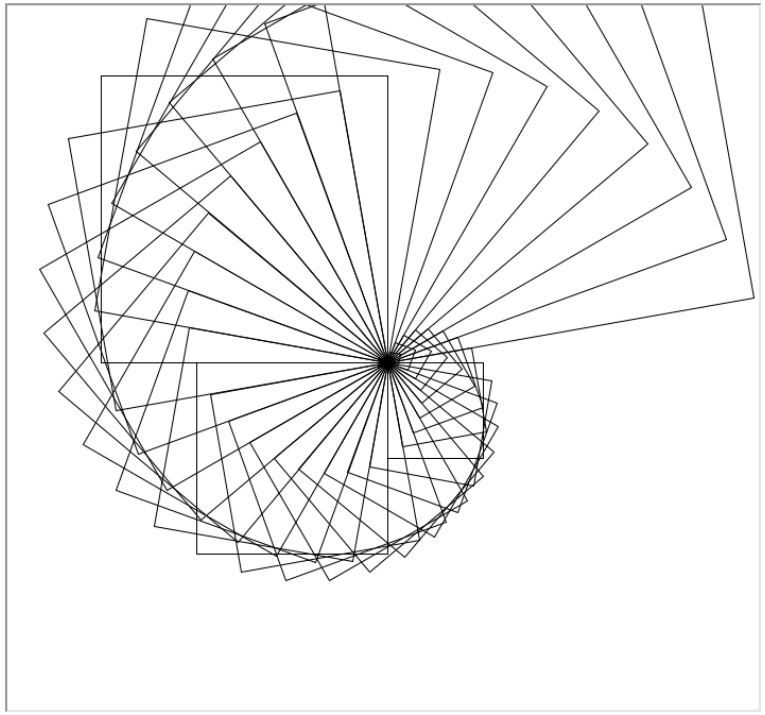
```
import turtle
turtle.reset()
turtle.speed('fastest')

# Draws a square
# Parameters: side, a positive integer.

def square(side):
  for i in range(4):
    turtle.forward(side)
    turtle.left(90)

# Let's use the new function

for i in range(36):
  square(10*i)
  turtle.right(10)
```



Improving Square Again?

- The function square now has a parameter size, but we are doing something that can be further parameterized
  - repeat 4 times
    - draw an edge, turn 360/4=90
- What about if we want a triangle? a polygon?

```
def square(side):
  for i in range(4):
    turtle.forward(side)
    turtle.left(90)
```

Hard-coded value

Hard-coded value

Polygon Function

- We can define a polygon function!
- What angle should we use to turn in the loop?

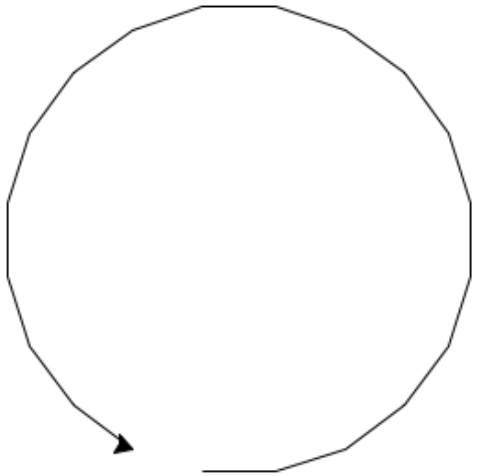
```
# Draws a 'n'-gon of size 'side'

def polygon(n, side):
  for i in range(n):
    turtle.forward(side)
    turtle.left(360/n)

# Use our polygon function to
# define a square function

def square(side):
  polygon(4, side)
```

What happens if I do `polygon(19, 30)`?



What's happening? How to fix?

Pacing and Understanding

How well did you understand today?

- **A** Too easy, this lecture is way below my abilities
- **B** Everything went at a good pace, and I am fine
- **C** Too fast, but I will catch up on my own
- **D** Too fast, and I need you to slow down
- **E** I really do not think I can handle this