Abstraction

Being abstract is something profoundly different from being vague... The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.
-- Edsger Dijkstra

Common mistakes from last week

- You only need to “import turtle” once, and you can put it at the top of your program
- Make sure your parentheses match!

Common mistakes from last week

- if emotion == ‘happy’ or ‘excited’:

  print ‘hooray!’
- The above doesn’t work because the condition is equivalent to:

  (emotion == ‘happy’) or (‘excited’)
- ‘excited’ is a truthy value, just like any non-empty string. Try the following in your workspace:

  if ‘some random string’:

  print ‘non-empty strings are truthy!’

Common mistakes from last week

- Instead, the following code will work:

```
if emotion == ‘happy’ or emotion == ‘excited’:
    print ‘hooray!’
```

Common mistakes from last week

- The following is an unnecessary amount of code:

```
value = raw_input(“How far to move?”)
winston.forward(int(value))
winston.right(90)
winston.forward(int(value))
winston.right(90)
winston.forward(int(value))
```

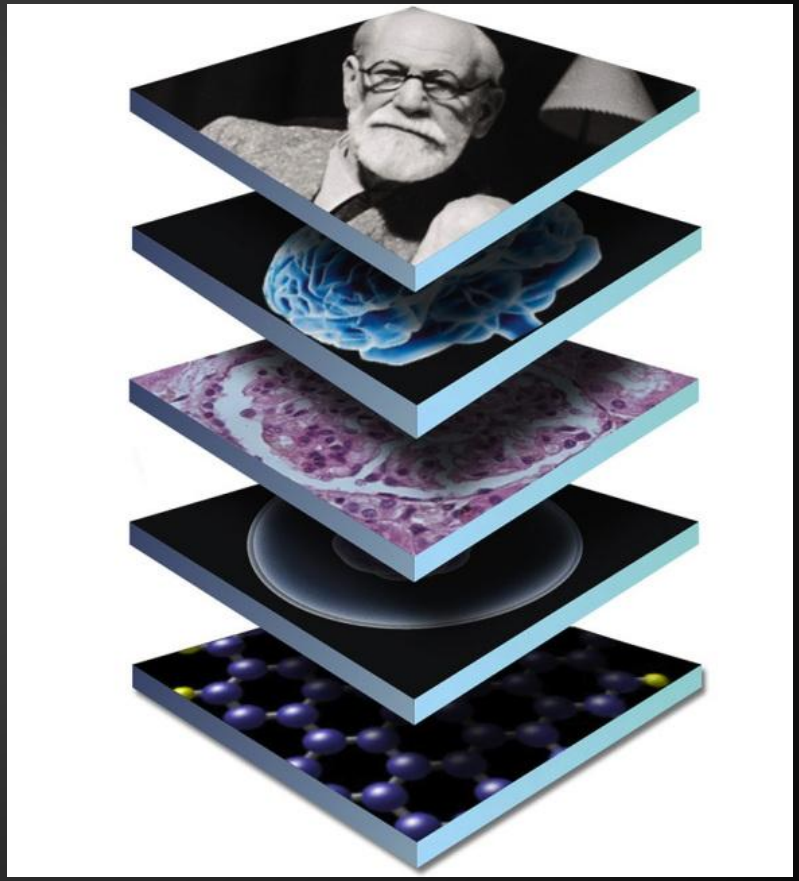
Common mistakes from last week

- Instead, make the variable value an integer:

```
value = int(raw_input("How far to move?"))
winston.forward(value)
winston.right(90)
winston.forward(value)
winston.right(90)
winston.forward(value)
```

What is abstraction?

- Programming is easy as long as the programs are small
- In a 10,000 line program, keeping track of all the small details at once becomes intractable
- Solution: chunking, or layering -- metaphors for abstraction

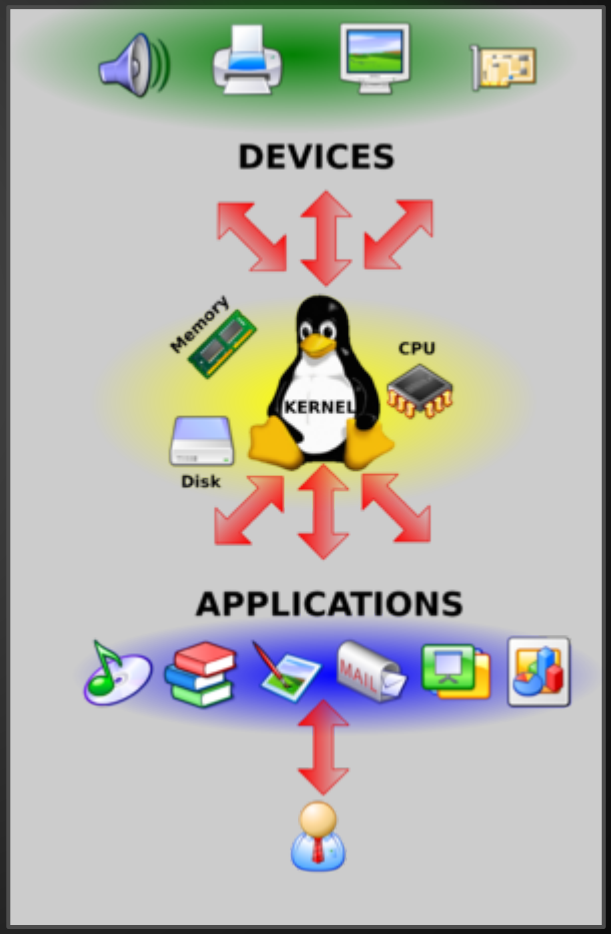


Classic example: consider a car

- What is a car made of?
  - nuts, bolts, metal rods, big metal blocks, rubber gaskets, plastic containers for fluids, wires, etc.
  - at a lower level: atoms, which are made of electrons, neutrons, protons, which are made of...
  - a mechanic thinks at a higher level: the engine, the fuel injectors, the brakes, the transmission, etc.
- Technological progress --> more abstraction
  - Automatic transmission makes driving really easy
  - Two pedals are “interface” or “abstraction barrier”
  - Underlying details can change without us having to worry about it

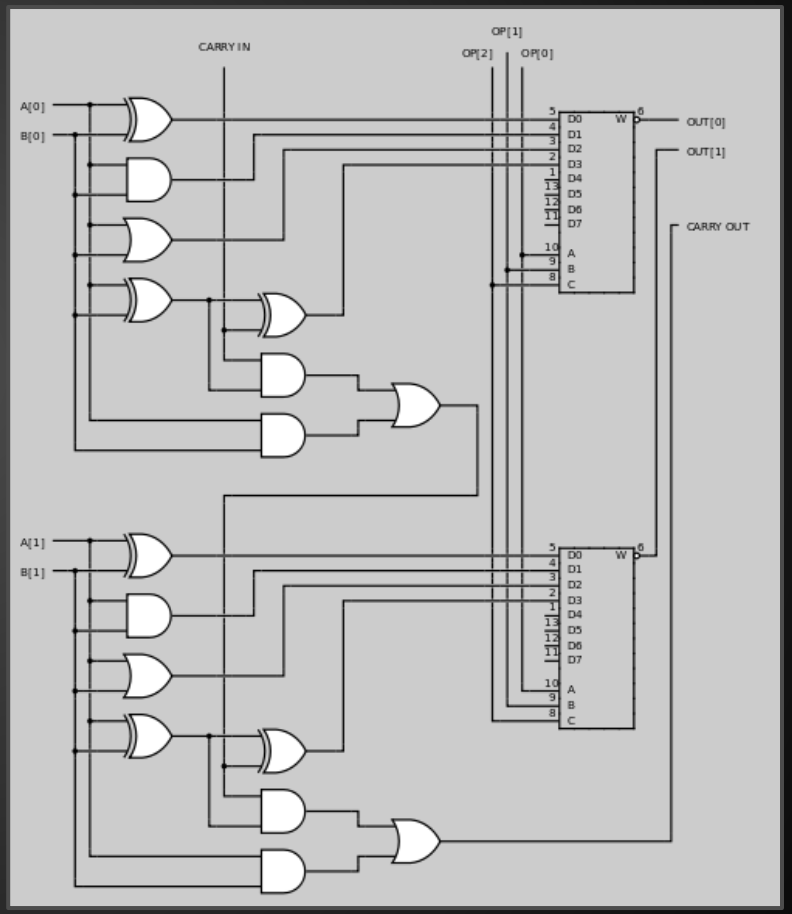
Computers are like onions

- Turtle graphics is an abstraction!
  - a collection of colored dots have to be redrawn a bunch of times on your screen
- Below Python: the operating system
  - usually comes in Mac, Windows, or Linux flavor
  - handles input/output, schedules processes, connects to WiFi, etc.



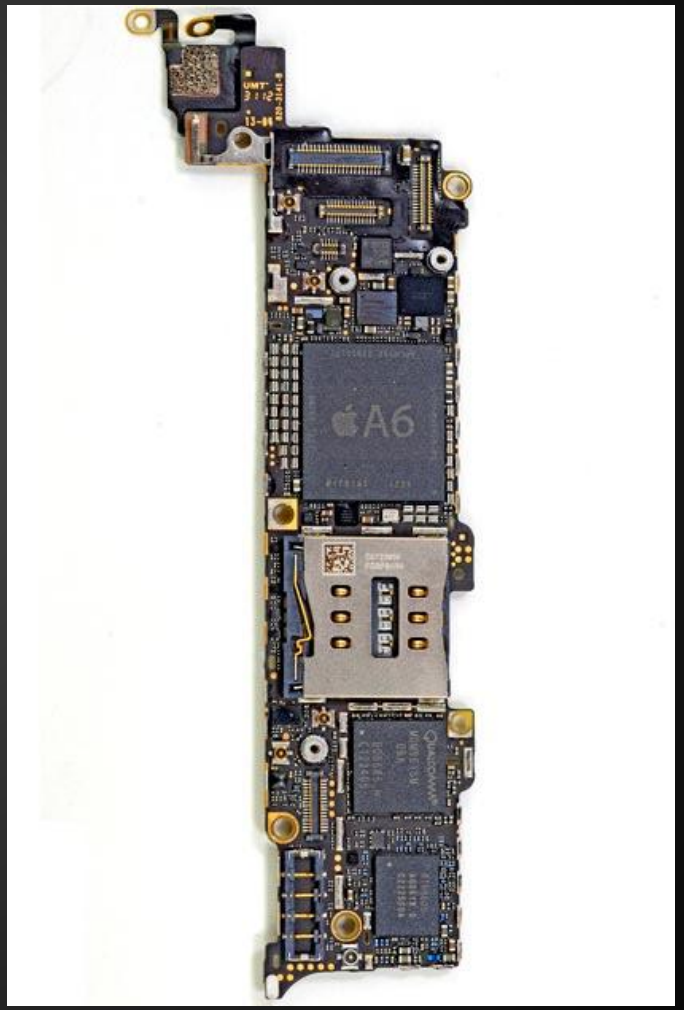
Computers are like onions

- Below the operating system: hardware
  - your processor has a central processing unit (CPU), memory (RAM), hard drive, etc.
  - abstractly represented as wires connected by logic gates
  - digital designers come up with schematics for all these things



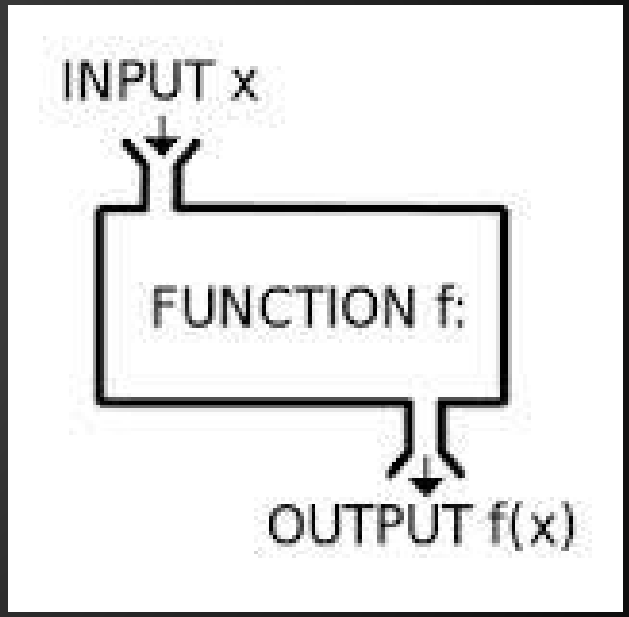
Computers are like onions

- Bottom level: physics
  - a transistor is a little pipe that controls electron flow, made out of silicon
  - new Intel chips have over a billion transistors in them!
  - spaces between transistors around 30 nm at the moment -- have to deal with quantum effects



Functions!

- Functions in Python allow us to abstract away multiple instructions into a single command
- You've been using them already: turtle.forward(), raw_input(), int(), etc.
- We say we're "calling" a function when we write its name followed by parentheses



Defining our own functions

```
def triangle():
    winston.forward(20)
    winston.left(60)
    winston.forward(20)
    winston.left(60)
    winston.forward(20)

triangle()
winston.forward(50)
triangle()
```

```
def square():
    winston.forward(20)
    winston.left(90)
    winston.forward(20)
    winston.left(90)
    winston.forward(20)
    winston.left(90)
    winston.forward(20)

square()
```

Functions can have arguments

```
def triangle(size):
    winston.forward(size)
    winston.left(60)
    winston.forward(size)
    winston.left(60)
    winston.forward(size)

triangle(40)
triangle(50)
```

```
def triangle(s, col):
    winston.color(col)
    winston.forward(s)
    winston.left(60)
    winston.forward(s)
    winston.left(60)
    winston.forward(s)

triangle(40, ‘red’)
triangle(50, ‘blue’)
```

Arguments can be optional

```
def triangle(s, col='red'):
    winston.color(col)
    winston.forward(s)
    winston.left(60)
    winston.forward(s)
    winston.left(60)
    winston.forward(s)

triangle(40)
triangle(50, 'blue')
```

Functions can call other functions!

```
def move_turn(dist, ang):
    winston.forward(dist)
    winston.left(ang)

def triangle(size):
    move_turn(size, 60)
    move_turn(size, 60)
    move_turn(size, 60)

triangle(30)
```

```
def square(size):
    move_turn(size, 90)
    move_turn(size, 90)
    move_turn(size, 90)
    move_turn(size, 90)

square(50)
```

Functions can return a value

```
def calc_angle(sides):
    total = 180 * (sides - 2)
    return total / sides

def triangle(size):
    ang = calc_angle(3)
    move_turn(size, ang)
    move_turn(size, ang)
    move_turn(size, ang)
triangle(30)
```

```
def square(size):
    ang = calc_angle(4)
    move_turn(size, ang)
    move_turn(size, ang)
    move_turn(size, ang)
    move_turn(size, ang)

square(50)
```

Questions?