Buffalo Nickel Moment

Show the random selector program first.

A. Scope of names

Scope – where things are visible (things = names)

Global vrs. Local
Class vrs. Instance

Global variables and global functions exist outside of any other functions or classes.

Local variables are only visible inside functions.
Parameters are always local.

Inside a function...

If you use a name, Python assumes it is global and looks for it outside the function.

If you assign to a name, Python assumes it is a local and creates a new local version.

If you want to assign to a global variable, you must first declare it as a global.

```python
names = ["Mark", "Sally", "Kathy"]
def silly(s):
    if s in names:
        print ("We know", s)

names = ["Mark", "Sally", "Kathy"]
def silly(s):
    names = ["weird", "stuff", s]

names = ["Mark", "Sally", "Kathy"]
def silly(s):
    global names
    names = ["weird", "stuff", s]

names = ["Mark", "Sally", "Kathy"]
def silly(s):
    names[0] = s
```

Since names is used in silly but not assigned to, Python looks for names outside, and it uses the global variable.

Since you assign to names inside the function, Python thinks names is a local. So the names inside is NOT the same as names outside, and no change occurs.

Here you are assigning but you have to tell Python that the names you are assigning to inside the function is the same as the one outside.

Here you are not assigning to names itself, but are actually changing the existing value in the variable, so global is not needed. Python assumes you mean the global names.
B. Object Oriented programming Part 1: evolution of ideas

Arrays were the first composite objects. Parallel arrays were the first form of quasi-database. Objects are **homogeneous** in arrays (all the same) at least the same size. Nobody talked about "objects" in 1957. They talked about values and arrays.

COBOL (1960) introduced the record. C programmers know it as "struct". The record is **heterogeneous**. You talked about fields, not positions, within the collection. And these fields were named.

```c
struct employee {
    char name[200];
    int yearBorn;
    double salary;
    int married;  // Boolean
    char race;
};

struct employee fulltimers[1000];  // array of structs

fulltimers[0].yearBorn = 1957;
fulltimers[1].salary = 100573.44;
```

Of course to make the mini-database, you'll need an array of these records or structs.

Python's dictionary is a generalization of a struct. In C, you associate a name (which is a string) to a value inside a struct (see above.) In Python, you can do the same (although the syntax is different, don't use the dot notation.) But in Python, the "name" can be any kind of object, including numbers, strings, sets, tuples, etc!

Simula-67 (created by Ole-Johan Dahl and Kristen Nygaard in the 1960s in Oslo, Norway) was used for simulation. Introduced so many concepts that are in all our languages today. Amazing! It was the first object oriented programming lang.

Think of a class as a struct with methods added to it. These methods often work on the fields of the struct:

```c
struct employee {
    char name[200];
    int yearBorn;
    double salary;
    int married;  // Boolean
    char race;

    void getRaise(double howMuchPct) {
        this.salary = this.salary + this.salary * howMuchPct;
    }
};
```
\[
\frac{3}{4} + \frac{1}{4} = \frac{4}{4} \quad \frac{3}{4} \times \frac{2}{3} = \frac{6}{12}
\]

def add(self, otherFrac):
    if self.den != otherFrac.den:
        blowup()
    self.num += otherFrac.num

f = Fraction(3, 4)
g = Fraction(1, 4)
f.add(g)
print(f.num, "/", f.den, sep=""
we write **classes**
but we create & use **objects**

A class is a new type.
" " " " the shape of all the objects made from it.

**Human**

-> **characteristics**

instance variables - peculiar to an object
class variables - common to all objects

**immutable**

**mutable**

max life expectancy

class Fraction:
" " " " This class implements...

def _init_(self):
    pass

def _init_(self, numerator, denominator):
    self.num = numerator
    self.den = denominator

f = Fraction (3,4)
global code

def functions inside functions
inner func disappear
upon return
just like a local var

scripting languages
bash Bourne-again shell
perl
PowerShell
Matlab
JavaScript ≠ Java
PHP
Ruby

compiled languages
C FORTRAN 1957
Java
C#
C++
### CSC 112 Recitation sections

<table>
<thead>
<tr>
<th>Time</th>
<th>Sunday</th>
<th>Mon</th>
<th>Tues</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2 PM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-3 PM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-4 PM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Session 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Session 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-5 PM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Session 2</td>
<td></td>
</tr>
<tr>
<td>5-6 PM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-7 PM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instructions: mark 2 boxes, not on the same day, during which time you could meet for one hour to hold the CSC 112 recitation. Put "1" into your first, most favorite choice. Put "2" into your second most favorite choice. Only mark the white boxes, not the gray boxes.

---

**Quiz 1:**
- lists, sets, dicts
- Ch. 8 — on your own
- Ch. 9 — first official Zybooks homework