# CS 4501/6501: In-class 6
# ISP Coverage Criteria

28-Sep-2017

## Names:

---

**Purpose**: Practice input space testing, apply ISP coverage criteria
**Instruction**: Work with your neighbors in groups. Consider a small web app, convert.php, that allows users to convert measurement. Design IDM for the app and then write tests that satsifty Base Choice Coverage. Note: Try to keep your partitioning simple and choose a small number of partitions and blocks.

1. List all of the input variables, including the state variables

```
14 textboxes: F, C, cm, in, ft, m, ...
2 buttons: Convert and Clear Form
Many browser features: back, forward, reload, entering URL directly, bookmark, ...
```

2. Define characteristics of the input variables. Make sure you cover all input variables

```
One possible solution would be
   * characteristics 1, input is a number or not
   * characteristics 2, number compare to 0
   * characteristics 3, form button is clicked
   * characteristics 4, browser feature is clicked
```

3. Partition the characteristics into blocks

```
   * characteristics 1, input is a number or not: [number, not number]
   * characteristics 2, number compare to 0: [neg, 0, pos]
   * characteristics 3, form button is clicked: [Convert, Clear Form]
        // Focusing on functionality of the app, we are interested in how the app
        // behaves in response to clicking the "Convert" button or the "Clear Form" button.
        // Note: not clicking the form button won't trigger the functionality of the app
        // (doesn't help us test the app).
        // Thus, "not click" is not considered as a block nor an input value here.

   * characteristics 4, browser feature is clicked: [click, not click]
```

4. Do all the partitions you design satisfy completeness and disjointness properties? If not, revise the partition(s)

```
 Satisfy both properties
```

5. Define values for each block

```
   * characteristics 1, [1, a]
   * characteristics 2, [-1, 0, 1]
   * characteristics 3, [Convert, Clear Form]
   * characteristics 4, [Back, not click]
```

6. Write a set of test requirements that satisfies Base Choice Coverage Then, identify any infeasible test requirements you might have.

```
First, let's decide the base choices
  * characteristics 1, [numer, not number], (base: number)
  * characteristics 2, number compare to 0: [neg, 0, pos], (base: pos)
  * characteristics 3, form button is clicked: [Convert, Clear Form], (base: convert)
  * characteristics 4, browser feature is clicked: [click, not click], (base: not click)
Thus, our base test requirement (tr) is
   tr 1: [number, pos, convert, not click]

Then, apply Base Choice Coverage (BCC) to derive the remaining test requirements (trs)
   tr 2: [number, pos, convert, click]
   tr 3: [number, pos, clear form, not click]
   tr 4: [number, neg, convert, not click]
   tr 5: [number, 0, convert, not click]
   tr 6: [not number, pos, convert, not click]

Identify any infeasible test requirement(s)
   tr 6 (block 2 of characteristic 1 and block 3 of characteristic 2 cannot be combined)

To deal with infeasible test requirement(s), there are two options
   option 1: discard it, or
   option 2: fix it
Typically, testers try to fix the infeasible test requirement(s).

Assume we want to fix the infeasible test requirement tr 6.
One way to fix this tr is to consider which aspect (between "not number" and "pos") is more important and then drop the less imp
Since other tests cover "number" and "pos," a reasonable option in this case is to cover "not number" aspect.
The fixed tr 6 would be [not number, convert, not click]
```

7. Write a test set (that satisfies Base Choice Coveage). Write your tests with the values from step 5. Reminder: each test case consists of test inputs and expected output.

```
test case 1: 1,1,1,1,1,1,1,1,1,1,1,1,1, convert  (base)
             Expected output (Post-state): results include
                    1.0 Fº = -17.22 Cº   1.0 Cº = 33.8 Fº   1.0 in = 2.54 cm   1.0 cm = 0.39 in
                    1.0 ft = 0.3 m       1.0 m = 3.28 ft    1.0 mi = 1.61 km   1.0 km = 0.62 mi
                    1.0 gal = 3.79 L     1.0 L = 0.26 gal   1.0 oz = 28.35 g   1.0 g = 0.04 oz
                    1.0 lb = 0.45 kg     1.0 kg = 2.21 lb
```

```
test case 2: 1,1,1,1,1,1,1,1,1,1,1,1,1, convert, back
             Expected output (Post-state): depend on business logic,
             for example, (i) clicking the browser back button should reset all previously entered data,
             or (ii) data entries previously entered should remain, ...

test case 3: 1,1,1,1,1,1,1,1,1,1,1,1,1, clear form
             Expected output (Post-state): all text fields are clear

test case 4: -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, convert
             Expected output (Post-state): results include
               -1.0 Fº = -18.33 Cº   -1.0 Cº = 30.2 Fº   -1.0 in = -2.54 cm    -1.0 cm = -0.39 in
               -1.0 ft = -0.3 m      -1.0 m = -3.28 ft   -1.0 mi = -1.61 km    -1.0 km = -0.62 mi
               -1.0 gal = -3.78 L    -1.0 L = -0.26 gal  -1.0 oz = -28.35 g    -1.0 g = -0.04 oz
               -1.0 lb = -0.45 kg    -1.0 kg = -2.2 lb

test case 5: 0,0,0,0,0,0,0,0,0,0,0,0,0 convert
             Expected output (Post-state): results include
                0.0 Fº = -17.78 Cº    0.0 Cº = 32.0 Fº    0.0 in = 0.0 cm    0.0 cm = 0.0 in
                0.0 ft = 0.0 m        0.0 m = 0.0 ft      0.0 mi = 0.0 km    0.0 km = 0.0 mi
                0.0 gal = 0.0 L       0.0 L = 0.0 gal     0.0 oz = 0.0 g     0.0 g = 0.0 oz
                0.0 lb = 0.0 kg       0.0 kg = 0.0 lb

test case 6: a,a,a,a,a,a,a,a,a,a,a,a,a convert
             Expected output (Post-state): depend on the implementation
                HTTP 500 Error is returned (for the Java servlet version, but not the php version)
```