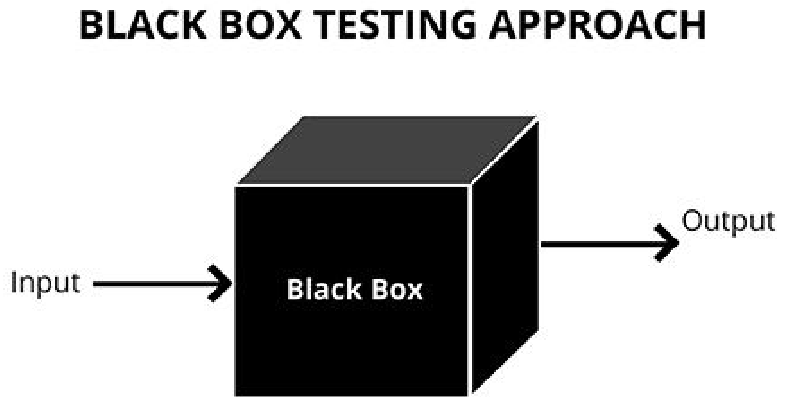
Lecture Notes
October 11, 2018

Black box testing focuses on inputs and outputs, without concern for how the software under test transforms the inputs into outputs

Most black box testing uses a test oracle to determine what is the expected output for the given input and to determine whether the actual output matches the expected output for that input



Since it is not feasible to test *all* inputs, the tester needs to select a practically small number of inputs that, collectively, are *most likely* to produce actual outputs that do not match the expected outputs - that is, the test cases most likely to reveal bugs

Some such inputs might be explicitly specified in the "conditions of satisfaction" from the customer

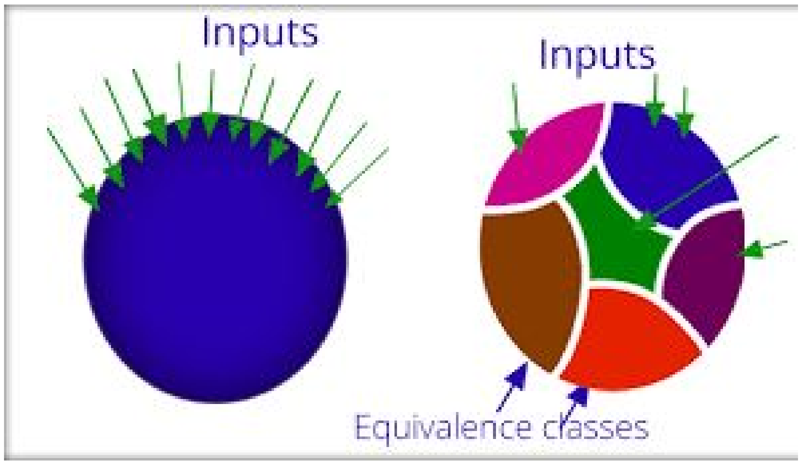
Is that all the test cases needed?

The customer knows the application domain, but may not know much about how the domain might be implemented in software

Can testers rely on the customer-provided conditions of satisfaction to consider everything that could possibly go wrong with the application software?

To err is human, but to really foul things up you need a computer.
Paul R. Ehrlich
BrainyQuote

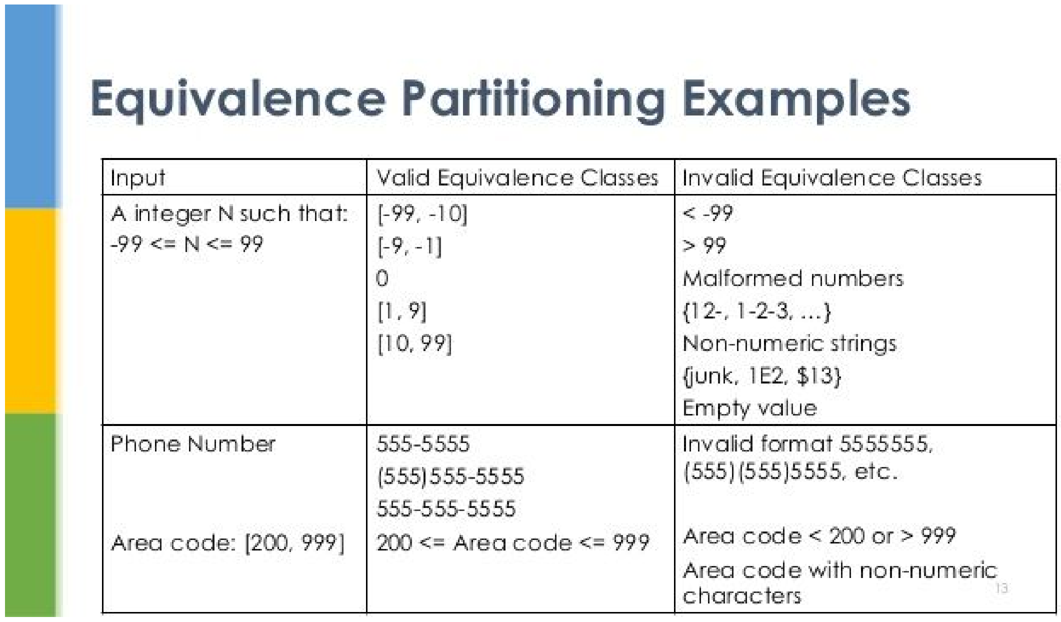
The simplest approach for representing all possible inputs to a software application with a practically small number of examples is to divide the input space into "*equivalence partitions*" (or equivalence classes) based on **both** domain knowledge and software engineering knowledge



An equivalence partition is a set of alternative input values where the software can reasonably be expected to behave equivalently (similarly) - this does **not** mean literally the same output for every input in the class

Instead it means that if the software under test behaves correctly on one example input chosen from that equivalence class, it is reasonable to believe it will behave correctly on all the other inputs in same class

Equivalence classes consider how inputs will be provided to the software as well as the application domain, with different classes for valid and invalid values



Equivalence Partitioning Examples

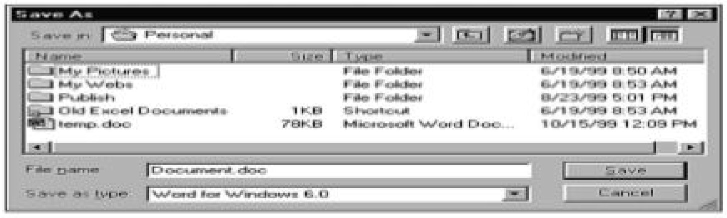
Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	< -99 > 99 Malformed numbers {12-, 1-2-3, …} Non-numeric strings {junk, 1E2, $13} Empty value
Phone Number	555-5555 (555)555-5555 555-555-5555	Invalid format 5555555, (555)(555)5555, etc.
Area code: [200, 999]	200 <= Area code <= 999	Area code < 200 or > 999 Area code with non-numeric characters

13

Why is it necessary to consider *invalid* inputs?

Example of Equivalence Partitions

- Windows filename can contain any characters except \/ : * ? " < > |.
- Filenames can have from 1 to 255 characters.
- Creating test cases for filenames:
  - Will have equivalence partitions for valid characters, invalid characters, valid length names, names that are too short and names that are too long.



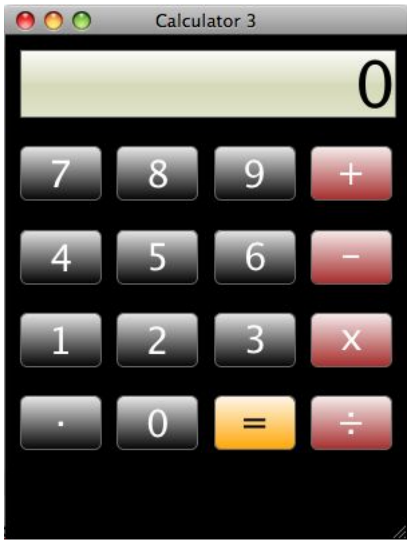
Save in: Personal

Name	Size	Type	Modified
My Pictures		File Folder	6/19/99 8:50 AM
My Webs		File Folder	6/19/99 8:53 AM
Publish		File Folder	8/23/99 5:01 PM
Old Excel Documents	1KB	Shortcut	6/19/99 8:53 AM
temp.doc	78KB	Microsoft Word Doc...	10/15/99 12:09 PM

File name: Document.doc

Save as type: Word for Windows 6.0

Consider a calculator implemented in software, with a GUI as shown (the user clicks the buttons with a mouse, but cannot enter text into the display)



In mathematics, the "+" operation takes two numerical operands

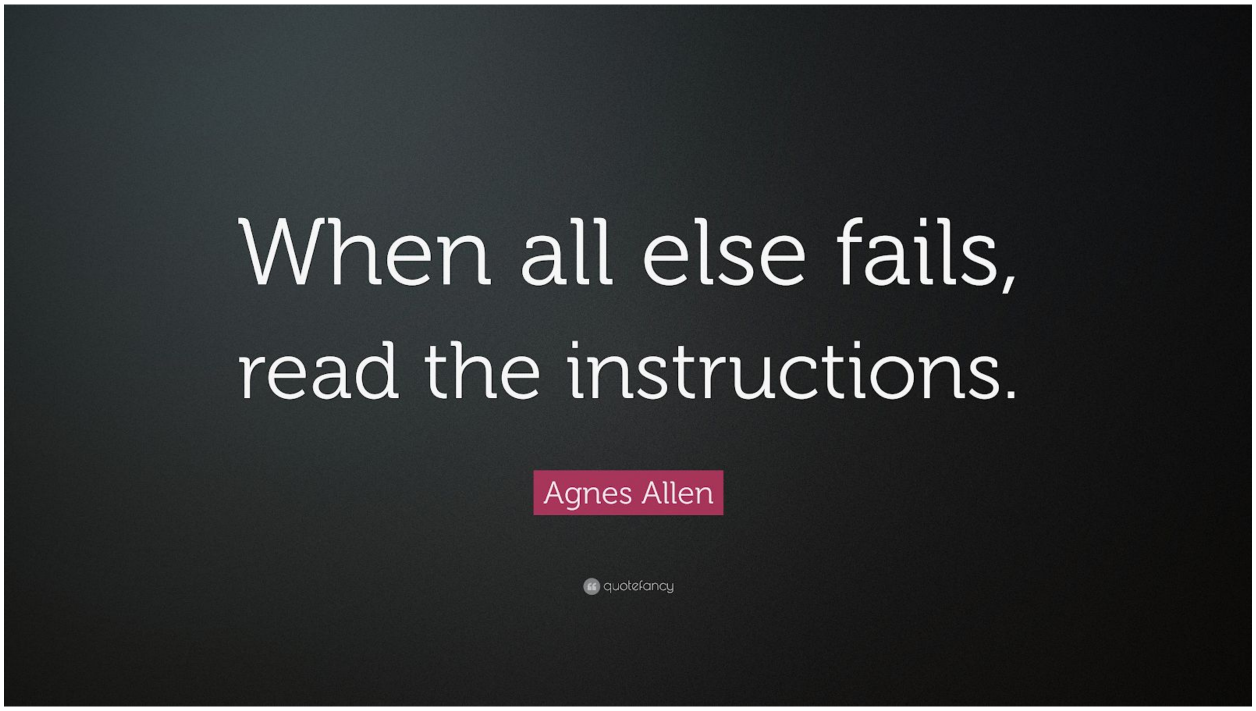
Say this software is tested by an external testing team separate from the developers. Based on *domain knowledge*, a tester might reasonably expect the "+" operation to behave similarly on all pairs of numbers < first operand, second operand >

So the tester tries an arbitrary pair of numbers: < 3.141592653, 42 >

The calculator flashes 00000000, now what?

Based on *software engineering knowledge*, the tester might realize the flashing 00000000 could indicate:

- An error message because the software cannot handle numbers with more than 8 digits
- An error message because the software cannot handle mixed operands, i.e., one operand is floating point and the other is integer
- A bug
- Something else


When all else fails,
read the instructions.
Agnes Allen
quotefancy

Let’s ignore the “.” key (planned for version 2.0) and assume the “+” operator is currently intended only for integers

A tester with software engineering knowledge, not just domain knowledge, might realize that a calculator implementation could behave differently for positive integers, zero, and negative integers

Thus she tests with five different equivalence classes:

- < pos int, pos int>
- < pos int, neg int >
- < pos int, 0 >
- < neg int, neg int >
- < 0, 0 >

Is this sufficient?

Although the "+" operator is mathematically commutative, that does not mean that the implementation (which could be buggy) is commutative

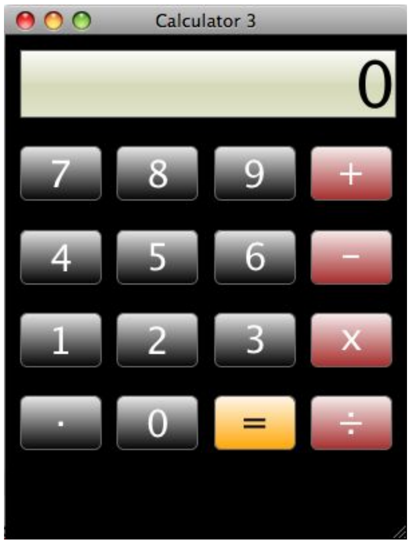
Better to test with:

- < pos int, pos int>
- < pos int, neg int >
- < pos int, 0 >
- < neg int, pos int >
- < neg int, neg int >
- < neg int, 0 >
- < 0, pos int >
- < 0, neg int >
- < 0, 0 >

Or, more generally, define equivalence partitions separately per parameter, and then extend to the cross-product

But notice the scaling problem with large numbers of parameters

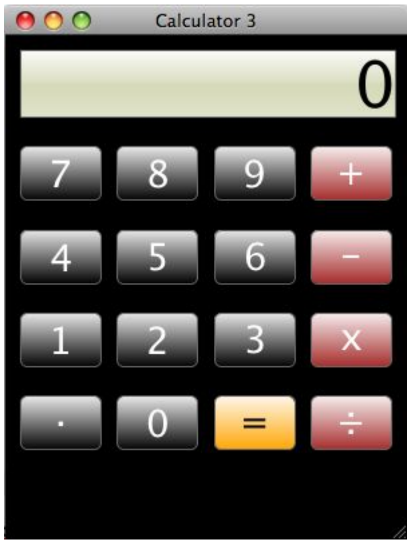
Are there other problems with independent equivalence partitions for each parameter?



Let's say the calculator is indeed intended to handle operands only up to 8 digits (or to any other fixed number of digits)

This means there is a MaxInt and a MinInt (negative), probably specific to what the calculator GUI was designed to display rather than MaxInt/MinInt of the programming language, compiler, operating system, etc.

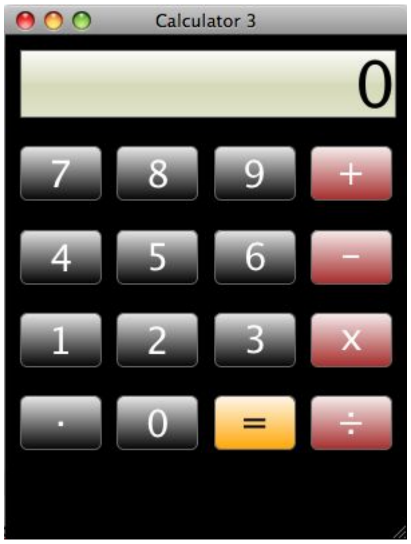
What additional tests are needed?



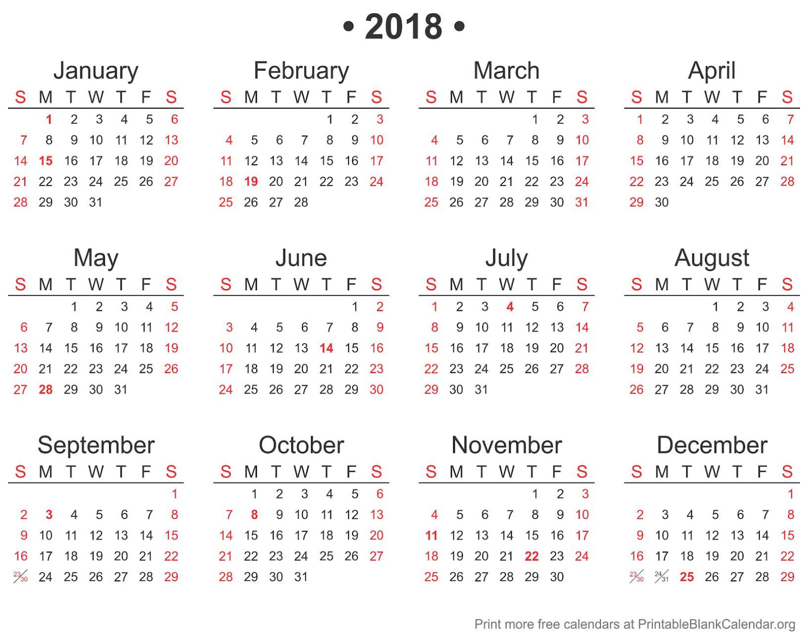
- < xxx, MaxInt >
- < MaxInt, xxx >
- < xxx, MinInt >
- < MinInt, xxx >
- < MaxInt, MinInt >
- < MinInt, MaxInt >

Any more?

What if the user could enter text, using a conventional keyboard, into the display window in addition to mousing the keys



Does that change the set of equivalence partitions that need to be tested?



• 2018 •

January

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	**15**	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

February

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	**19**	20	21	22	23	24
25	26	27	28			

March

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

April

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

May

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	**28**	29	30	31		

June

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	**14**	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

July

S	M	T	W	T	F	S
1	2	3	**4**	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

August

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

September

S	M	T	W	T	F	S
						1
2	**3**	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23/30	24	25	26	27	28	29

October

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	**8**	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

November

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	**22**	23	24
25	26	27	28	29	30	

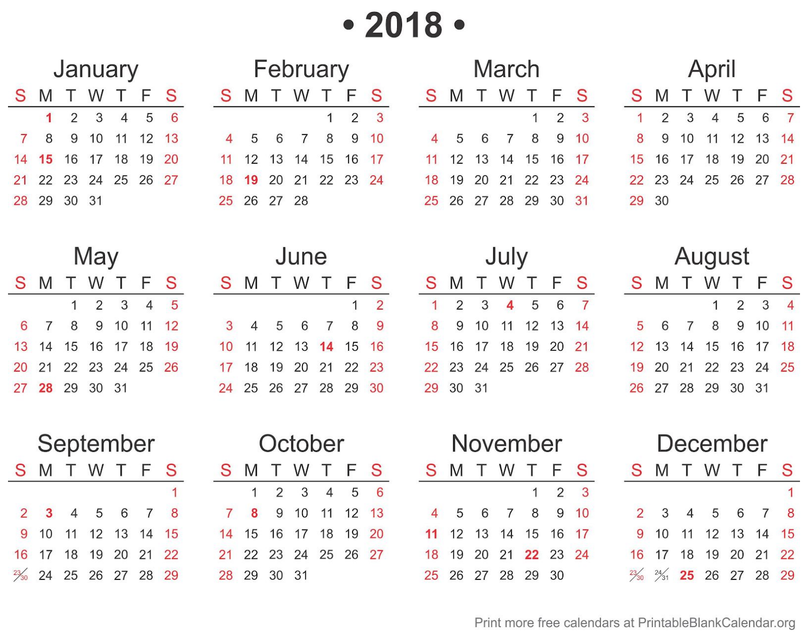
December

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23/30	24/31	**25**	26	27	28	29

Print more free calendars at PrintableBlankCalendar.org

Consider a calendar program that represents months as integers 1 to 12

What are the equivalence classes?



• 2018 •

January

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	**15**	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

February

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	**19**	20	21	22	23	24
25	26	27	28			

March

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

April

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

May

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	**28**	29	30	31		

June

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	**14**	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

July

S	M	T	W	T	F	S
1	2	3	**4**	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

August

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

September

S	M	T	W	T	F	S
						1
2	**3**	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23/30	24	25	26	27	28	29

October

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	**8**	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

November

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	**22**	23	24
25	26	27	28	29	30	

December

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23/30	24/31	**25**	26	27	28	29

Print more free calendars at PrintableBlankCalendar.org

Range 1..12 is one equivalence class, all valid inputs are in this class

Integers below range - zero and negative - is another equivalence class

Integers above range - 13 and higher - is another equivalence class. Why is this a different equivalence class than 0 and lower?

Non-integers represent one or more additional equivalence classes for invalid inputs

• 2018 •

January

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	**15**	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

February

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	**19**	20	21	22	23	24
25	26	27	28			

March

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

April

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

May

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	**28**	29	30	31		

June

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	**14**	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

July

S	M	T	W	T	F	S
1	2	3	**4**	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

August

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

September

S	M	T	W	T	F	S
						1
2	**3**	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23/30	24	25	26	27	28	29

October

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	**8**	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

November

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	**22**	23	24
25	26	27	28	29	30	

December

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23/30	24/31	**25**	26	27	28	29

What would be some good test inputs for test to pass?
(initial smoke test)

What would be some good test inputs for test to fail?
(most testing)

Consider another calendar program that represents months as strings, “January”, “February”, and so on

Instead of a *range* of valid inputs, there is a *set* of valid inputs

Non-members in the set form one invalid equivalence class, instead of below range and above range as invalid equivalence classes

Some non-members of the valid set may be validly formatted whereas others may not be does this matter?

What does formatting mean here?

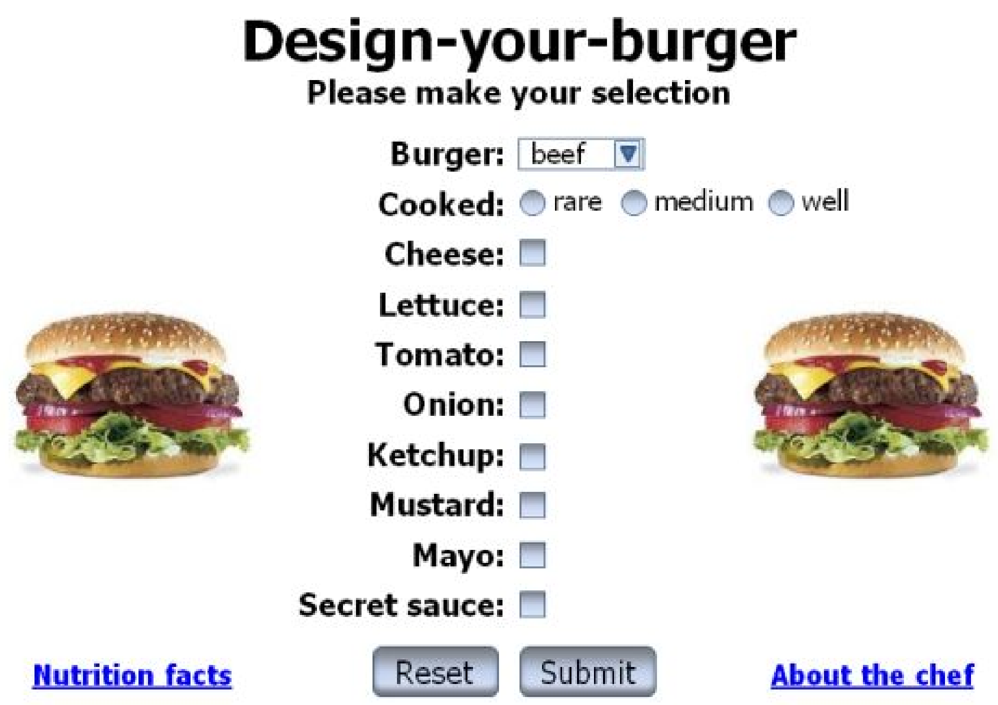
- Printing vs. non-printing characters
- Upper vs. lower case
- Space, tab, newline characters (whitespace)
- Any characters with special meaning to the application (or to the implementation, e.g., language, shell, SQL)
- Null/empty string
- Any string above maximum buffer size (the buffer is the data storage for reading the string input)

What would be some good test inputs for test to pass?

What would be some good test inputs for test to fail?

Say the same calendar program that represents months as strings, “January”, “February”, and so on also allows *unambiguous* abbreviations, e.g., “F”, “Fe”, “Feb”

What are the equivalence partitions?



Design-your-burger

Please make your selection

Burger: beef ▼

Cooked: ○ rare ○ medium ○ well

Cheese: ☐

Lettuce: ☐

Tomato: ☐

Onion: ☐

Ketchup: ☐

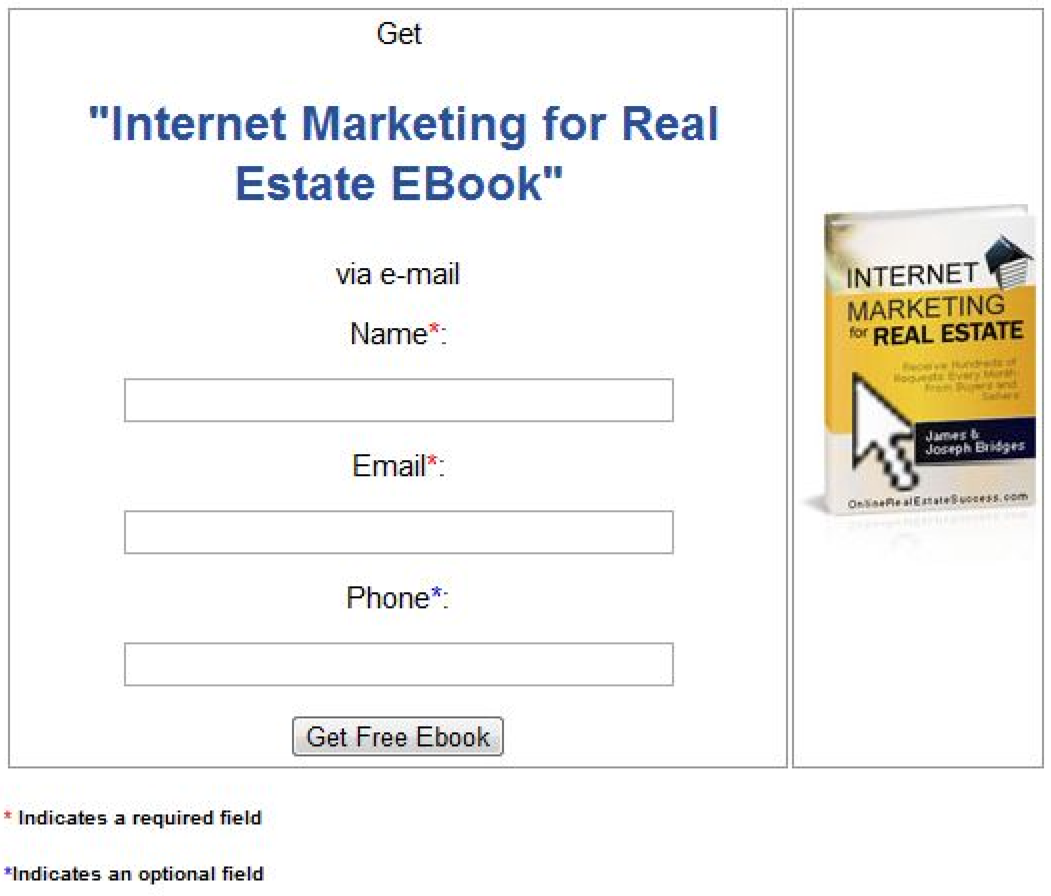
Mustard: ☐

Mayo: ☐

Secret sauce: ☐

What are the equivalence partitions?

Equivalence classes also need to address required vs. optional inputs (with or without default values)


Get

"Internet Marketing for Real Estate EBook"

via e-mail

Name*:

Email*:

Phone*:

Get Free Ebook
INTERNET MARKETING for REAL ESTATE
James & Joseph Bridges
OnlineRealEstateSuccess.com

* Indicates a required field

*Indicates an optional field

Equivalence classes may cut across multiple inputs:

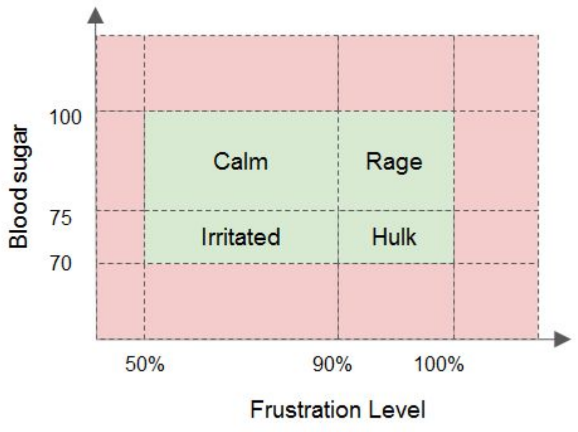
- Sometimes the value of one input restricts appropriate values for other inputs
- For example, date of birth and age should be consistent (valid) or not (invalid)

Distinct parameters may be related to each other in forming equivalence classes

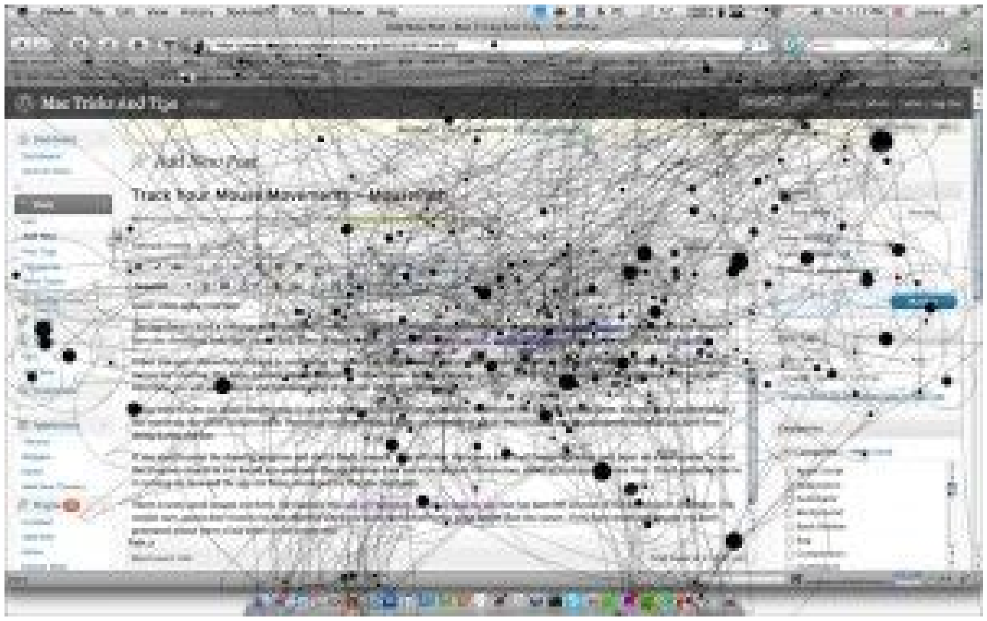
Let's say we had a function that took two variables (blood sugar level and level of frustration) to calculate "mood"

Assume valid inputs are
- Blood sugar 70-100
- Frustration is 50% .. 100%

For 50 <= F <= 90
  75 <= BS <= 100: Calm
  BS < 75: Irritated
For F>90
  75 <= BS <= 100: Rage
  BS < 75: Hulk Smash

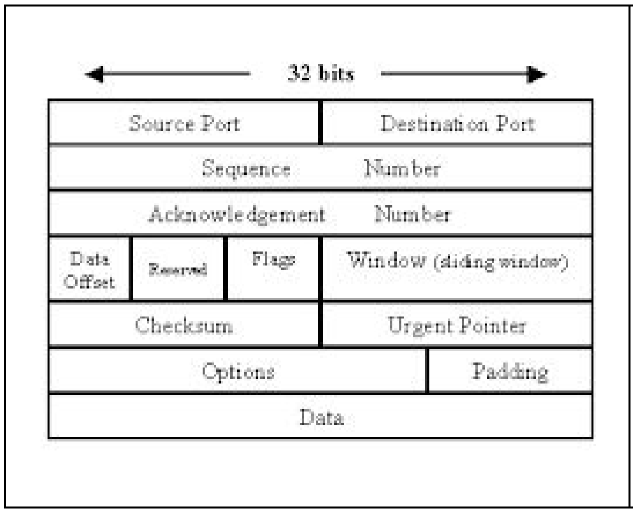


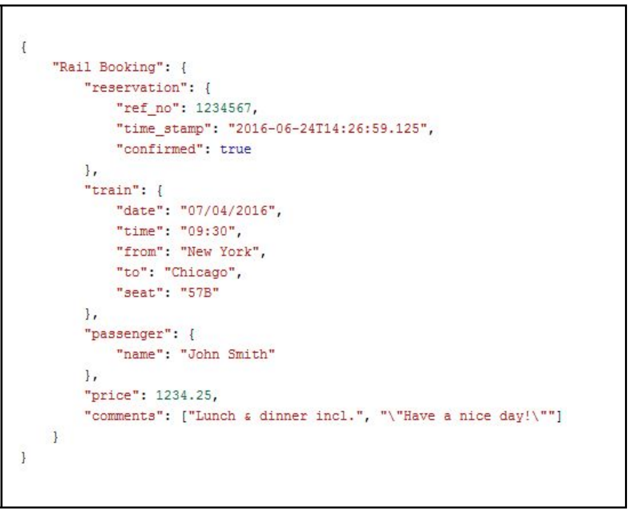
For GUI applications, inputs may also be mouse movements, locations or clicks, manual (touch-sensitive) gesture, camera images, gps coordinates, etc.



What might be some valid and invalid equivalence classes for mouse activity?

System-to-system inputs are likely to be specially formatted, e.g., network packets, JSON




```
{
    "Rail Booking": {
        "reservation": {
            "ref_no": 1234567,
            "time_stamp": "2016-06-24T14:26:59.125",
            "confirmed": true
        },
        "train": {
            "date": "07/04/2016",
            "time": "09:30",
            "from": "New York",
            "to": "Chicago",
            "seat": "57B"
        },
        "passenger": {
            "name": "John Smith"
        },
        "price": 1234.25,
        "comments": ["Lunch & dinner incl.", "\"Have a nice day!\""]
    }
}
```

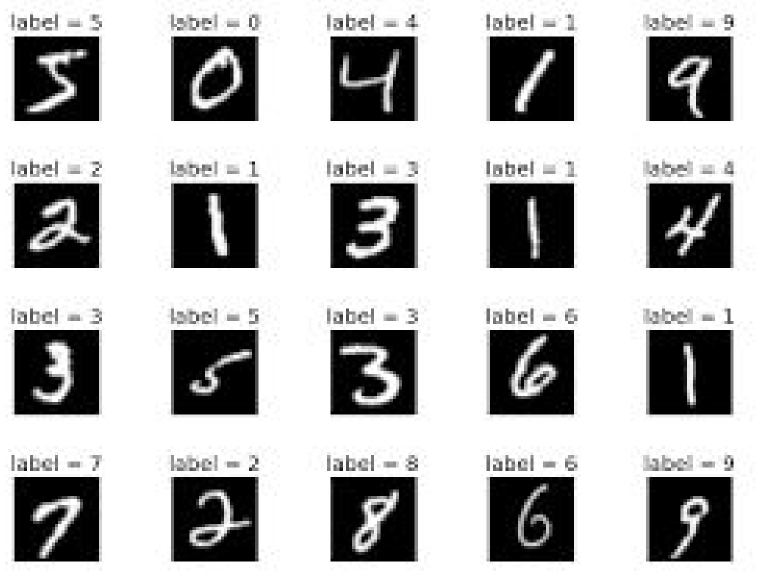
Each field has its own equivalence partitions, as well as higher-level correctly formatted vs. not (or multiple correct formats if alternatives supported)

File inputs can:

- Exist or not
- Be below min or above max size
- Be readable, writeable, executable by current user (permissions)
- Correct, corrupted, garbled, etc. data and metadata formats - images, audio, video, …

Files containing "data sets" often correspond to a table (rows and columns)

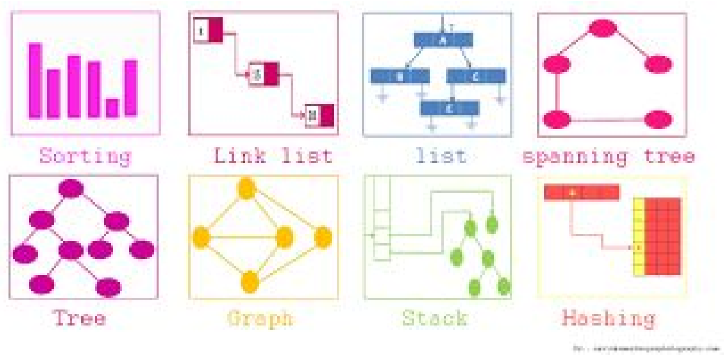
This shows part of a well-known dataset for deep learning algorithms (MNIST)



What are the equivalence classes?

At the full system level, inputs are generally going to be numbers, strings (text), tuples (over network or from devices), and files

At the unit testing level, inputs/outputs can be arbitrary data structures, so equivalence classes can be more complex - and inputs more complicated to construct

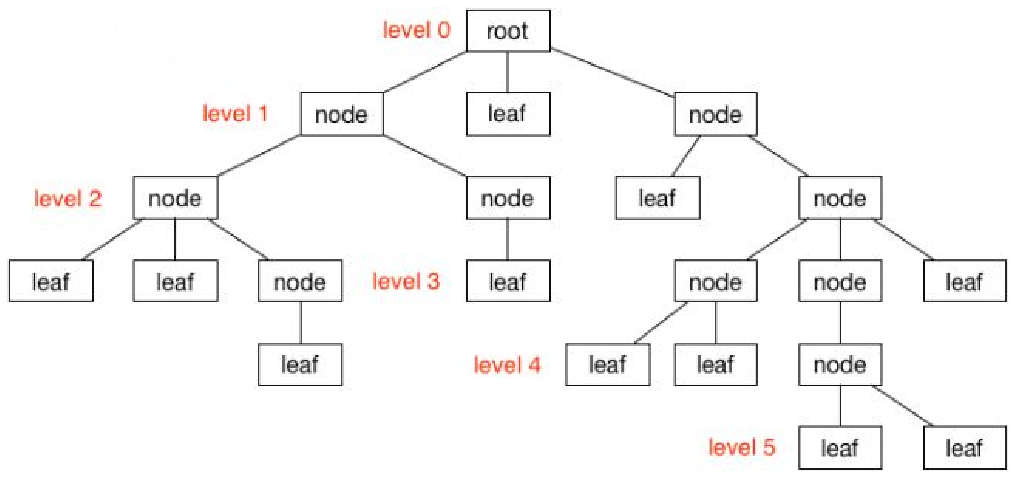


Container data structures with content elements:

- In or not in container
- Empty or full container (or min/max number of elements)
- Duplicate elements may or may not be allowed
- Ordered or organized correctly, or not

Specific kinds of containers, e.g., tree - root node, interior node, leaf node, null tree, tree with exactly one node, “full” tree, balanced vs. unbalanced, various specialty trees - and graphs

How would test cases construct root, node and leaf objects to use as inputs?



It is possible to construct sample inputs at unit level that could never actually be supplied when the unit is executed within the full program

If those inputs find bugs, are these false positives or true positives?

In-class exercise if time permits:

- Group together in teams of ~4 with whoever is sitting near you
- Recall the hypothetical drone that flies around the classroom, and uses its sensors to determine which students are “paying attention” and which are not
- Assume you can test your application using a configurable classroom simulator, which supplies all sensor and wifi data (these are the test inputs)
- What are the equivalence classes for valid inputs?
- What are the equivalence classes for invalid inputs?

Ten minutes

Volunteers present their equivalence partitions

Read for Tuesday:
textbook, chapter 5 Criteria-Based Test Design

Next pair assignment, Bug Hunt due next Tuesday, 10:10am.

Next team assignment First Iteration due Thursday, October 30, 11:59pm. Follow-up team assignment First Iteration Demo due Thursday, November 8, 11:59pm.

Midterm Individual Assessment will become available after class next Tuesday, October 16, due a week later on Tuesday, October 23, 11:59pm. This is a **hard** deadline.