Equivalence partitioning

- Divides the input data of a software unit into partitions of equivalent data assuming that all the conditions in one partition will be treated in the same way by the software.

- We need to test only one condition from each partition.

- If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others.

- Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work.

Example Program

A savings account in a bank has a different rate of interest depending on the balance in the account. The rate chart is as following:

- ✔ 0.5% rate of interest is given if the balance in the account is in the range of $0 to $1000
- ✔ 0.75% rate of interest is given if the balance in the account is in the range of $1000 to $10000
- ✔ 1% rate of interest is given if the balance in the account is above $10000

Invalid data	**Partition 2**	**Partition 3**	**Partition 4**
Balance <0	0 ≤ Balance <1000	1000 ≤ Balance< 10000	10000 ≤ Balance
Interest = undefined	Interest = 0.5%	Interest = 0.75%	Interest = 1.0%

How not to test?

float calculateSavingsInterest(float balance)

```
float interestCalc= calculateSavingsInterest(0);
assertEquals(0.0, interestCalc);
float interestCalc= calculateSavingsInterest(500.0);
assertEquals(2.5, interestCalc);
float interestCalc= calculateSavingsInterest(1000.0);
assertEquals(7.5, interestCalc);
float interestCalc= calculateSavingsInterest(2000.0);
assertEquals(15.0, interestCalc);
… … …
float interestCalc= calculateSavingsInterest(5000.0);
assertEquals(37.5, interestCalc);
```

How many partitions tested?

How to test?

float calculateSavingsInterest(float balance)

1
```
try{
    float interestCalc= calculateSavingsInterest(-1.0);
    fail("Exception expected")
}catch(Exception ex){}
```

2
```
float interestCalc= calculateSavingsInterest(100.0);
assertEquals(0.50, interestCalc);
```

3
```
float interestCalc= calculateSavingsInterest(2000.0);
assertEquals(15.0, interestCalc);
```

4
```
float interestCalc= calculateSavingsInterest(20000.0);
assertEquals(200.0, interestCalc);
```

Boundary value analysis

- Boundary value analysis is based on testing at the boundaries between partitions
- More application **errors occur at the boundaries** of input domain.
- Used to identify errors at boundaries rather than finding those exist in center of input domain.

Invalid data	Partition 2	Partition 3	Partition 4
Balance <0	0 ≤ Balance <1000	1000 ≤ Balance <10000	10000 ≤ Balance
Interest = undefined	Interest = 0.5%	Interest = 0.75%	Interest = 1.0%
-0.01	0, 0.01, 999.99	1000, 9999.99	10000

Positive testing

- Positive testing validates that an application will work on giving valid input data.

- Verifies that the application is NOT showing error when it is not supposed to

What is your phone number? | 6184536032

Negative testing

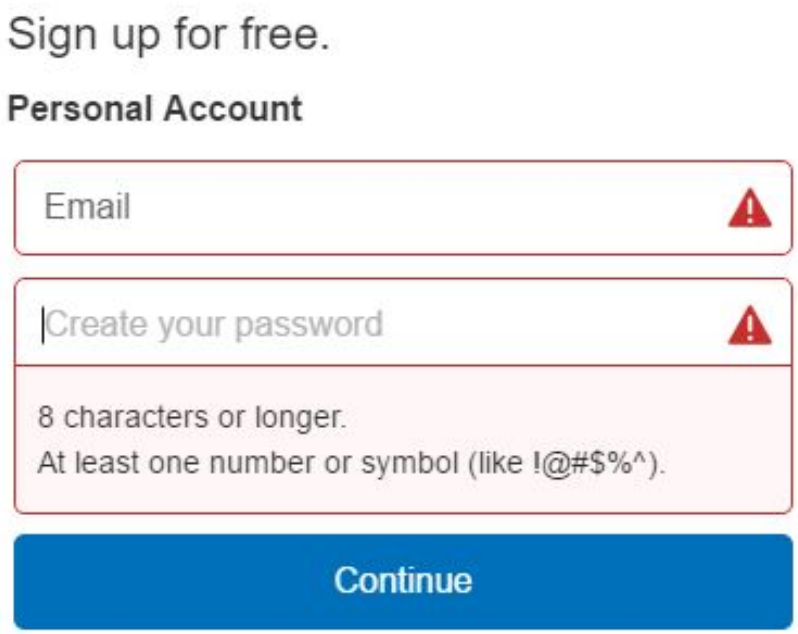
- Performed on the system by providing **invalid data as input**.
- Checks whether an application behaves as expected with the negative input.
- Ensures that your application can gracefully handle invalid input or unexpected user behavior.

What is your phone number?

xyZ#53()032

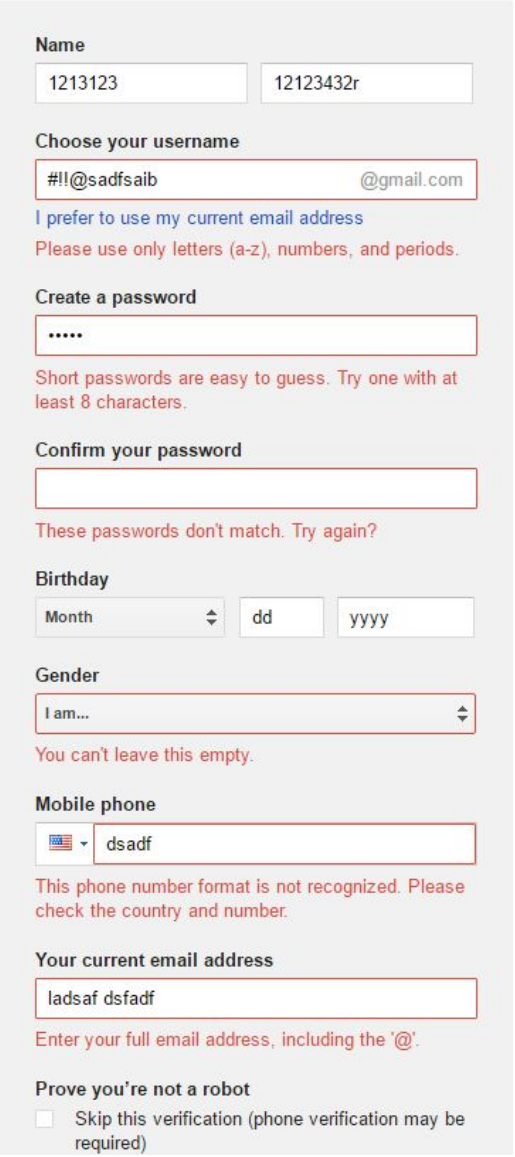
Typical Negative Testing Scenarios

- Not populating required fields



Correspondence between data and field types

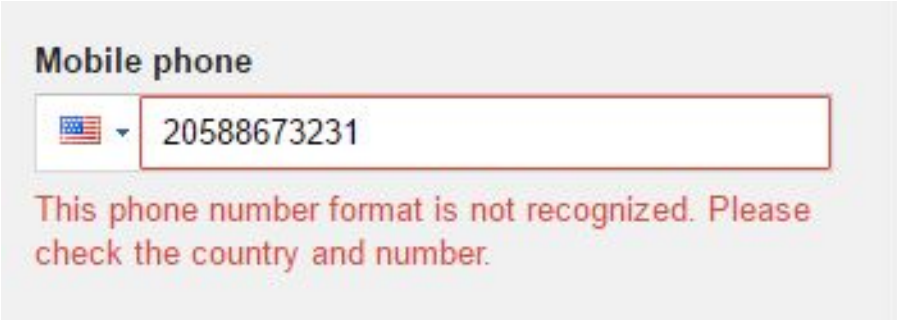
Populating fields with incorrect data type



Allowed number of characters

- Some applications or webpages may have set maximum / minimum values or number of characters for fields

- Create a test that enters more characters into the field than is allowed.



Allowed data bounds and limits

- Applications can use input fields that accept data in a certain range

- Create a negative test that enters a value smaller than the lower bound or greater than the upper bound of the specified field

Please enter day? -1

Please enter month? 13

Out of range values

- Create test exceeding the bounds of the data types

- Integer range:
  - -32,768 to 32,767 [2 bytes]
  - -2,147,483,648 to 2,147,483,647 [4 bytes]

- Float range:
  - -3.4E+38 to +3.4E+38

Reasonable data

- Check application with unreasonable input data
  - Name with 300 characters
  - Human Age: 200 years

Web session testing

- Can we access any URL without logging in?
  - Record some of the URL when you were logged in, try those again after you are logged out