

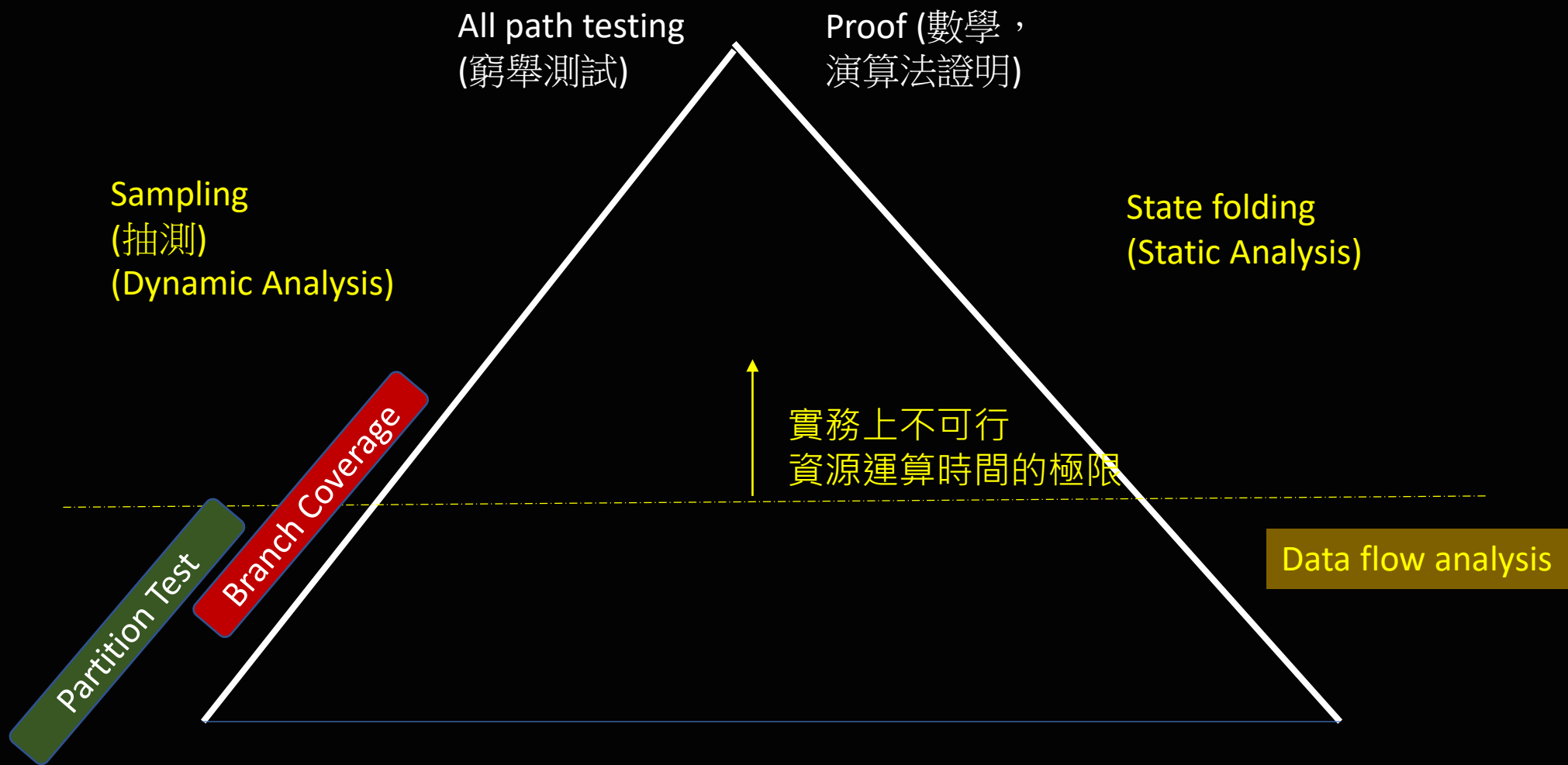
5. Partition Testing and Boundary (Edge) testing

學習目標

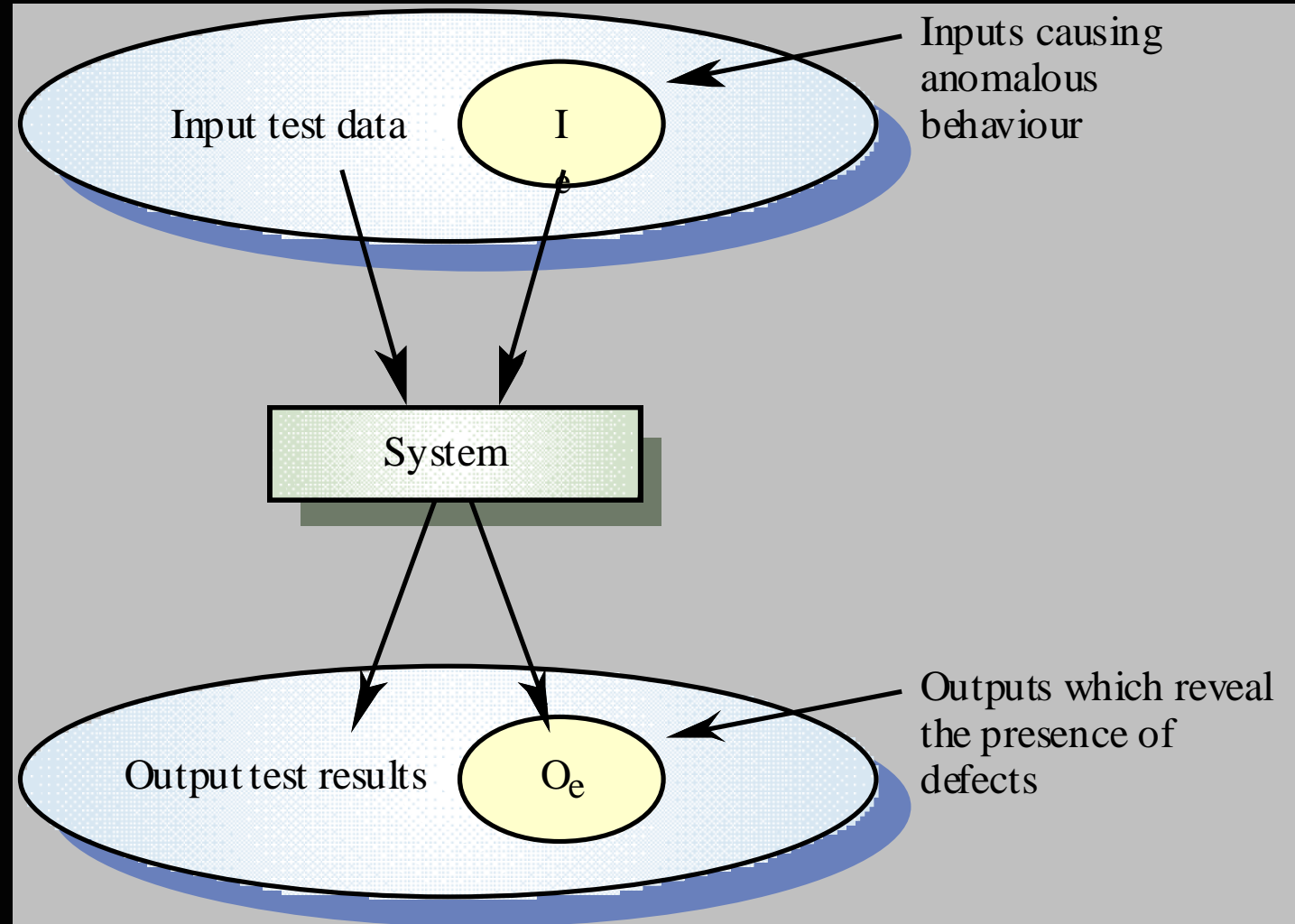
了解什麼是 partition testing

partition testing 與 code coverage 的關係是什麼

Partition Tests



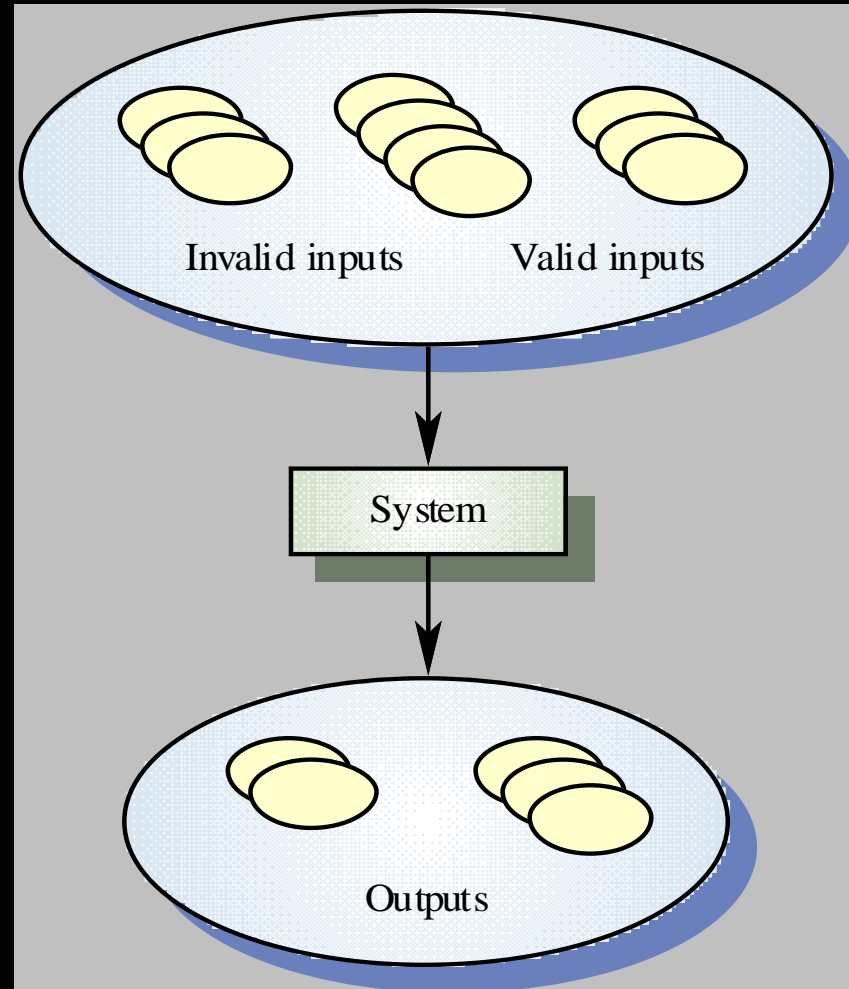
人類最古老的測試理論 Partition Tests



Equivalence partitioning (partition testing) (抽樣的技巧?)

- Input data and output results often fall into different classes where all members of a class are related
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each partition

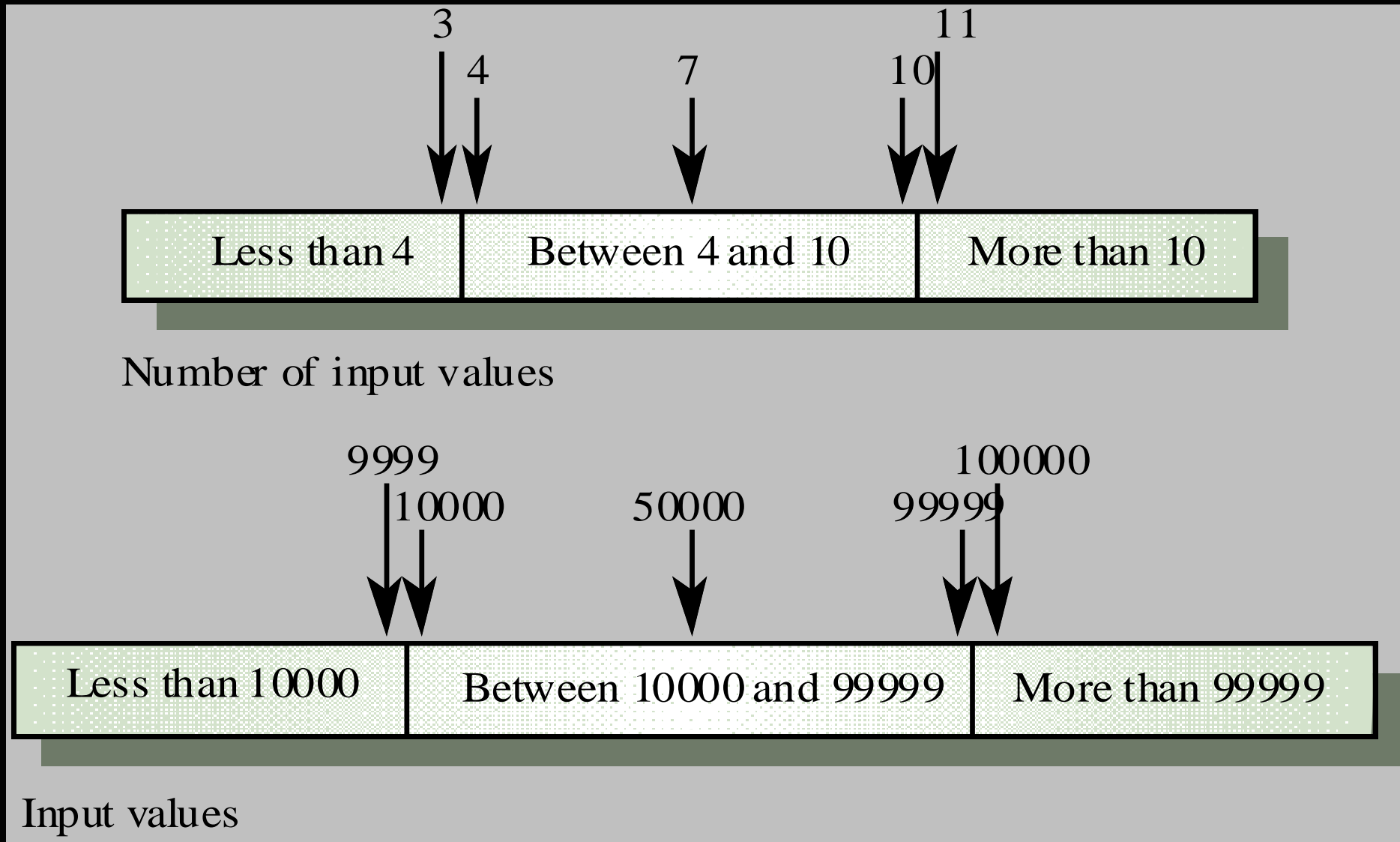
Equivalence partitioning



How to compute partitions in xUnit tests?

- White box testing
- 基本上你可以從 source code 的 if-then-else 做個 macro partition，通常這麼做你也完成了大架構的 statement code coverage
- 如果有 loop 那就得從主控制 loop 的變數與記憶體著手 (e.g., sorting problems)，從那一邊你大概能得到 partitions 的基本訊息。基本上同一個 partition 的 test data，都會讓你的程式碼走過一樣的 path (這其實是種本能，如果你過去 coding 經驗已經昇華到了某個境界)
- 每一個 partition, 挑一個測試案例

Equivalence partitions and boundary data selection



Partition Tests and Coverage 的關係

- Partition Testing (分割測試) 和程式碼覆蓋率是旨在提升軟體測試的品質和效果的相關概念，但它們著重於測試的不同方面。
- Partition tests，也被稱為equivalence partition，是一種測試抽樣技術，將輸入空間進行分割或一組預期會呈現相似行為的輸入值。其理念是，如果一個分割內的測試案例揭示了一個缺陷，那麼同一個分割內的其他測試案例也很可能揭示相同的缺陷。分割測試有助於減少測試中的測試資料準備，確保全面覆蓋不同的輸入情境。
- 程式碼覆蓋率是一個衡量程式在測試期間執行的程度的指標。它確定了在測試案例中哪些部分的程式碼已被覆蓋或執行。程式碼覆蓋率的目標是識別沒有被測試到的程式碼區域，並確保測試套件足夠全面，能夠執行不同的程式碼路徑、分支和語句。
- 分割測試和程式碼覆蓋率之間的關係在於，分割測試可以幫助引導創建達到更高程式碼覆蓋率的測試案例。通過識別具有相似行為的分割或輸入值群組，分割測試可以確保為每個分割創建代表性的測試案例，涵蓋不同的輸入情境，並增加執行不同程式碼路徑的機會。

Lab 5.1: (15-20 mins)

**請下載 `classifyTriangle.*`，分析，
並且寫出你設計的 `partition test`**

Code 解説

```
1 def classify_triangle(side1, side2, side3):
2     if side1 == side2 == side3:
3         return "Equilateral"
4     elif side1 == side2 or side1 == side3 or side2 == side3:
5         return "Isosceles"
6     else:
7         return "Scalene"
8
```

Discussion (隱藏答案)

Partition 與 code coverage 開始分離

- 你應該發現，其實只要三個 test cases 就可以達到 statements coverage 100%
- 但是 partition test 的話，你應該要有 5 個以上

**Lab 5.2: 請下載 bubble_sort.* ,
分析，並且寫出你設計的
partition tests**

Code 説明

```
1  def bubble_sort(arr):
2      n = len(arr)
3
4      # Traverse through all array elements
5      for i in range(n):
6          # Last i elements are already in place
7          for j in range(0, n - i - 1):
8              # Swap adjacent elements if they are in the wrong order
9              if arr[j] > arr[j + 1]:
10                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
11
12     return arr
13
```


Discussion (隱藏答案)

Boundary(Edge) Tests

Types of Boundary testing

- **Boundary tests** (邊界測試案例) 是一種軟體測試技巧，旨在測試輸入值的邊界條件。邊界測試案例專注於處理輸入的極端情況，通常是在有效範圍的邊界值上進行測試。它的目的是驗證系統是否正確處理極端情況，例如最小和最大值，以及邊界值周圍的情況。以下是一些常見的邊界測試案例類型：
- **最小值和最大值**：測試輸入值的最小和最大合法範圍。這包括驗證系統是否能夠正確處理最小和最大可能的輸入值。
- **範圍邊界**：測試輸入值在有效範圍邊界上的情況。這包括驗證系統在接近邊界值時的行為，例如接近最小或最大值的輸入。
- **邊界之間**：測試輸入值在有效範圍邊界之間的情況。這包括驗證系統在邊界值之間的值上的行為。
- **精度和小數位數**：測試浮點數或具有特定精度要求的數值輸入的邊界情況。這包括驗證系統對於小數位數、四捨五入和近似值的處理。
- **字符串長度**：測試字符串輸入在最小和最大長度限制下的邊界情況。這包括驗證系統在最短和最長輸入上的處理方式。

更深奧的 Boundary Tests

- **邊界交叉**：測試輸入值在不同邊界條件交叉的情況。這包括同時測試多個邊界條件，以驗證系統在這些情況下的行為。
- **邊界組合**：測試多個邊界條件組合的情況。這包括測試不同邊界條件的組合，以確保系統在多個邊界條件同時存在時的正確處理。
- **邊界延伸**：測試接近邊界值但超出有效範圍的輸入情況。這有助於驗證系統在處理超出範圍的輸入時是否能夠適當地檢測和處理。
- **邊界順序**：測試邊界值以不同順序出現的情況。這可以確保系統在不同邊界值順序的情況下仍能正確運作。
- **特殊字符**：測試輸入值包含特殊字符或特殊情況的情況。這可以涵蓋例如空值、空白字符、控制字符等特殊情況。
- **時間和日期**：測試與時間和日期相關的輸入值的邊界情況，例如閏年、月份的最後一天等。

An example of Boundary Combinations

```
function calculateShippingCost(weight, distance) {  
    // Calculate shipping cost based on weight and distance  
    // ...  
  
    // Return the calculated shipping cost  
    return shippingCost;  
}
```

```
describe('calculateShippingCost', () => {
  test('Minimum weight and minimum distance', () => {
    const weight = 0;
    const distance = 0;
    const expectedCost = calculateShippingCost(weight, distance);
    expect(expectedCost).toBe(/* expected shipping cost */);
  });

  test('Maximum weight and maximum distance', () => {
    const weight = MAX_WEIGHT;
    const distance = MAX_DISTANCE;
    const expectedCost = calculateShippingCost(weight, distance);
    expect(expectedCost).toBe(/* expected shipping cost */);
  });

  test('Minimum weight and maximum distance', () => {
    const weight = 0;
    const distance = MAX_DISTANCE;
    const expectedCost = calculateShippingCost(weight, distance);
    expect(expectedCost).toBe(/* expected shipping cost */);
  });

  test('Maximum weight and minimum distance', () => {
    const weight = MAX_WEIGHT;
    const distance = 0;
    const expectedCost = calculateShippingCost(weight, distance);
    expect(expectedCost).toBe(/* expected shipping cost */);
  });

  test('Average weight and average distance', () => {
    const weight = AVG_WEIGHT;
    const distance = AVG_DISTANCE;
    const expectedCost = calculateShippingCost(weight, distance);
    expect(expectedCost).toBe(/* expected shipping cost */);
  });
});
```

厲害的 QA 通常會很厲害在 Boundary Tests

研究顯示，BUGS 被發現的地方集中在 boundary



**Lab 5.3: (10-15 mins) 請幫忙
bubble_sort 這個老掉牙的程式碼撰寫出
你心目中的 boundary test cases 。**

Discussion (隱藏答案)

Negative Tests

Negative Test Cases

- **無效輸入**：測試案例中，系統接收到的輸入不符合預期格式、資料類型或範圍。例如，將字串傳遞給需要數字的地方，或在要求非空值的地方提供空值。
- **邊界條件**：測試案例中，輸入值位於有效範圍的極端端點。這些測試案例檢查系統在面對最小或最大值時是否能正確運作。例如，使用數值輸入的最小和最大可能值進行測試。
- **錯誤處理**：模擬錯誤狀況的測試案例，例如網絡故障、數據庫連接問題或檔案系統錯誤。這些測試案例驗證系統是否能正確處理和恢復錯誤，顯示有意義的錯誤訊息或執行錯誤日誌記錄。
- **異常情況**：設計以觸發和驗證異常處理機制的測試案例。這些場景涉及到可能發生異常的情況，例如除以零、空引用或記憶體不足等。目標是確保系統能夠優雅地處理異常，並且不會崩潰或進入不穩定狀態。
- **負面業務規則**：檢查系統對於違反業務規則或約束條件的處理方式的測試案例。例如，嘗試執行不符合業務邏輯的操作或提交無效表單。
- **安全漏洞**：評估系統對於安全威脅（如注入攻擊、跨站腳本（XSS）或未經授權的訪問嘗試）的抵禦能力的測試案例。這些測試案例旨在發現

Invalid
input

System

Error

negative test

Valid
input

System

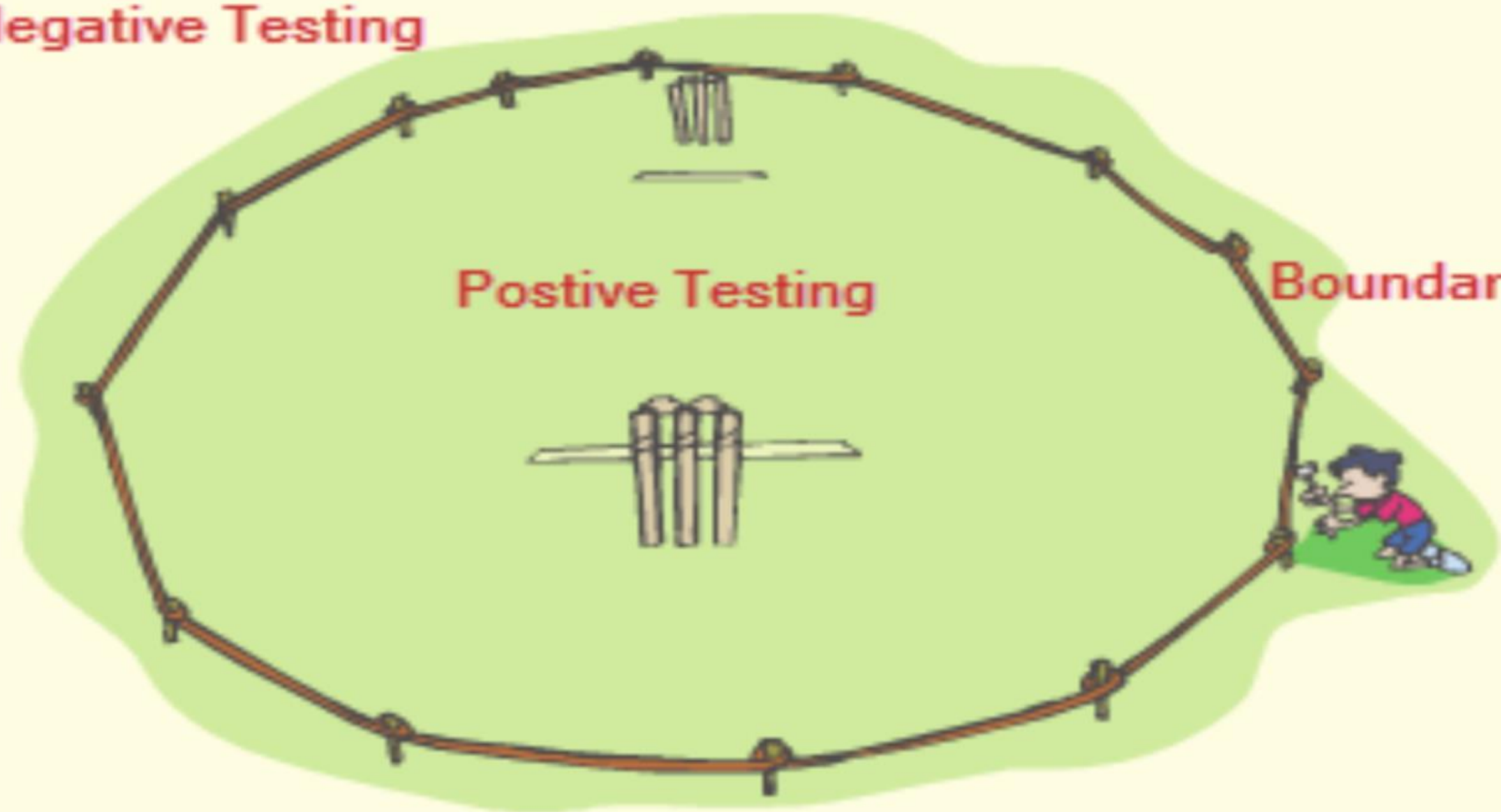
Expected
result

positive test

Negative Testing

Positive Testing

Boundary



Difference Between Positive and Negative Testing

Sno	Positive Testing (Valid)	Negative Testing (Invalid)
1	Positive Testing means testing the application or system by giving valid data.	Negative Testing means testing the application or system by giving invalid data.
2	Positive Testing is done by keeping positive point of view for example checking the mobile number field by giving numbers only like 9999999999.	Negative Testing is done by keeping negative point of view for example checking the mobile number field by giving numbers and alphabets like 99999abcde.
3	It is always done to verify the known set of Test Conditions .	It is always done to break the project and product with unknown set of Test Conditions.
4	This Testing checks how the product and project behave by providing valid set of data.	This Testing covers those scenarios for which the product is not designed and coded by providing invalid set of data.

Negative Test Cases

可以看出你的社會化程度以及成熟經驗值

- 沒錯，學生通常不會去測 negative test cases，因為學生不會去寫例外處理
- 剛出社會你知道要寫例外處理，不能因為 negative inputs 整個系統就停機與當機
- 例外處理的撰寫是工程師的事，而你既然寫了例外處理，那當然要想辦法觸發，也就是要測
- 隨著你的成熟經驗值越來越高，你應該知道建造一個可用的軟體例外處理很多很麻煩。研究顯示，例外處理的程式碼可以佔到 1/3 強
- 再進階一點，你所處的是一個多人的團隊。人越多，你就會發現不做例外處理(當然也不做 negative tests) 的人，簡直就是老鼠屎。


如果你的團隊發很多時間在整合過程中 debug



Example	Positive	False positive	Negative	False negative
Diabetes diagnosis	Bad	Bad	Good	Really bad!
Defect analysis	Bad	Bad	Good	Really bad!
Lottery win	Good	Really bad!	Bad	Bad
Code coverage	Good	Really bad!	Bad	Bad

Lab 5.4: (20 mins)
請下載 findNumbers.* 。

一個簡單到不能再簡單的程式

 findNumbers.py > ...

```
1  def find_numbers(numbers):
2      result = []
3      for num in numbers:
4          if num > 0:
5              result.append("positive")
6          elif num < 0:
7              result.append("negative")
8          else:
9              result.append("zero")
10     return result
11
```

Lab 5.4:

寫測試與寫程式有時候是個互動/互相成長的過程

1. 請先撰寫 Boundary/Negative test cases
2. 對 findNumbers.py 進行必要的改寫，來應付這些 test cases

請繳交

1. test_findNumbers.py
2. 修正成長之後的 findNumbers.py

Discussion (隱藏)