w05-Lec2

Syntax, Semantics Testing and Debugging

Assembled for 204111

by Kittipitch Kuptavanich

Syntax vs Semantics

- ในภาษา programming แต่ละภาษาจะประกอบด้วย
  - **Primitive Constructs,**
  - **A Syntax,**
  - **A static semantics,**
  - **A Semantics.**
- เราจะอธิบาย **Concept** โดยเปรียบเทียบกับภาษาที่ใช้ในชีวิตประจำวัน

Syntax vs Semantics

- **Primitive Constructs**
  - ในทางภาษา
    - **Set of words: cat dog คน น้ำชา รถยนต์**
  - ในทาง **programming**
    - `3.0 x y + = ==` **operators, operands, keywords…**

      etc

Syntax vs Semantics [2]

- **Syntax**
  - ในทางภาษา = ไวยากรณ์:
    - เช่น น้ำฉันดื่ม ← ไม่เป็นประโยค

      (ต้องวาง *subject verb object*)
  - ในทาง **programming**
    - `4 5 == 2` `# incorrect syntax`
    - `y = 4 * 5;` `# correct syntax`

Syntax vs Semantics [2]

- **Static Semantics**
  - บอกว่าประโยคที่*ถูกต้อง*ตามหลัก grammar หรือ syntax นั้น มีความหมายหรือไม่
  - ในทางภาษา
    - เช่น ปลาดาวปลูกอากาศ # อะไรคือความหมาย??????
  - ในทาง programming
    - 4 / "abc" # Syntax ถูกต้อง `<literal> <operator> <literal>`
      # แต่ไม่สื่อความหมาย

Syntax vs Semantics [3]

- **Semantics หรือ Full Semantics**
  - ความหมายของ expression หรือ ประโยคที่ถูก syntax และ statics semantics
  - ในทางภาษา
    - ข้าวเย็นหมดแล้ว # ความหมายกำกวม
  - ในทาง Programming
    - อะไรคือความหมายของ expression นั้น ๆ
    - `bmi = weight / height * height`
      - ถูก syntax และถูก static semantics
      - แต่ให้ผลคำนวณที่ผิด
      - ควรเป็น `bmi =` _____

Summary on Syntax and Semantics

- **Syntax**
  - ถูกหลักไวยากรณ์ไหม?
- **Static Semantics**
  - สื่อความหมายไหม?
- **Semantics (หรือ Full Semantics)**
  - ความหมายที่สื่อคืออะไร?
    - (ตรงตามที่ต้องการไหม?)

Syntax vs Semantics [4]

สิ่งที่อาจเกิดขึ้นในกรณีโปรแกรมที่เขียนทำงานผิดพลาด

- โปรแกรม crash, หยุดทำงาน
  - ถ้าโชคร้ายคือต้อง restart เครื่อง
- อาจจะ run ต่อไปเรื่อย ๆ ไม่หยุด หรือไม่แน่ใจว่าจะใช้เวลาอีกนานแค่ไหน
- ทำงานจนสิ้นสุดโปรแกรม แล้วให้ผลการ run ที่อาจจะผิดหรือถูก แล้วแต่กรณี

- กรณีใดเป็นกรณีที่แย่ที่สุด?
- เพราะเหตุใด?

Syntax vs Semantics [5]

- Error ที่พบบ่อยที่สุดคือ Error ด้าน Syntax
  - อันตรายน้อยที่สุด และตรวจหาได้ไวที่สุด
- Compiler/Interpreter สามารถช่วยตรวจหา Static Semantic error ได้บ้าง
- ในขณะที่โดยมาก Full Semantics Error เราจำเป็นต้องตรวจหาเอง
  - Testing and Debugging

TESTING AND DEBUGGING

Testing and Debugging

- Testing
  - กระบวนการในการ run โปรแกรมเพื่อ*ทดสอบ*ว่าโปรแกรมทำงานตามที่ต้องการหรือไม่
- Debugging
  - กระบวนการในการ*แก้ไขโปรแกรม*ที่ทราบว่ามีการทำงานที่ไม่ตรงตามต้องการ
- ควรพิจารณาการออกแบบ โปรแกรมให้สะดวกต่อการทำ Testing และ Debugging
  - แบ่งส่วนออกเป็นฟังก์ชันหรือโมดูลย่อย ๆ

Testing

- จุดประสงค์เพื่อค้นหาข้อผิดพลาดในโปรแกรม
  - ไม่ใช่เพื่อพิสูจน์ว่าโปรแกรมนั้น ๆ ไม่มีข้อผิดพลาด

> *"No amount of experimentation can ever prove me right; a single experiment can prove me wrong."*
> - Albert Einstein

  - ถ้า Test แล้วไม่พบ bug
    - ไม่ได้แปลว่าไม่มี bug

Conducting Testing

- โดยปกติ การทำ testing จะประกอบด้วย 2 ช่วง
  - Unit Testing: ทดสอบการทำงานของ*หน่วยย่อย* (เช่น ฟังก์ชัน) ของ code
  - ของ code Integration Testing: ทดสอบการทำงานของ*ระบบโดยรวม*
- การทำ Testing แบ่งเป็น 2 ประเภทหลัก ๆ
  - Glass-box (White-box) Testing: การสร้าง test ผ่านการพิจารณา code
  - Black-box Testing: การสร้าง test ผ่านการพิจารณา specification

Black-Box Testing

Black-box Testing คือการสร้าง test case จาก specification (ข้อกำหนด) ของโปรแกรม/ฟังก์ชัน

- ตัวอย่างเช่น

```
def sqrt(x, epsilon):
```

  - Spec:
    - ให้ `x (x >= 0)` และ `epsilon (epsilon > 0)` เป็น `float`
    - Return `result` ที่
      - `x - epsilon ≤ result * result ≤ x + epsilon`

Black-Box Testing [2]

- จาก specification เราอาจพิจารณา test 2 กรณี x > 0 และ x = 0
  - ไม่เพียงพอ
- สำหรับการคำนวณในลักษณะนี้ ควร test จำนวนที่เล็กหรือใหญ่มาก ๆ ด้วย
- 4 แถวแรกคือ กรณีปกติทั่ว ๆ ไป
  - x ที่เป็น perfect square
  - x ที่เป็น 0
  - x ที่น้อยกว่า 1
  - และ x ที่มีรากเป็นจำนวนอตรรกยะ
- หากโปรแกรมทำงานพลาดในกรณีนี้ แสดงว่ามี bug อยู่ใน code

x	epsilon
0.0	0.0001
25.0	0.0001
0.5	0.0001
2.0	0.0001
2.0	1.0/2.0**64.0
1.0/2.0**64	1.0/2.0**64.0
2.0**64.0	1.0/2.0**64.0
1.0/2.0**64.0	2.0**64.0
2.0**64.0	2.0**64.0

Black-Box Testing [3]

- ในแถวถัด ๆ มาคือ x และ *epsilon* ที่มีขนาดเล็กหรือใหญ่มาก ๆ
- หากโปรแกรมทำงานผิดพลาด
  - อาจมี bug ใน code
  - หรืออาจต้องแก้ spec
    - เช่นหาก *epsilon* มีขนาดเล็กมาก การหารากที่สองด้วยการประมาณค่าในลักษณะนี้อาจไม่สามารถทำได้

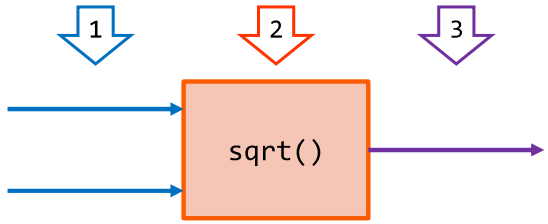
x	epsilon
0.0	0.0001
25.0	0.0001
0.5	0.0001
2.0	0.0001
2.0	1.0/2.0**64.0
1.0/2.0**64	1.0/2.0**64.0
2.0**64.0	1.0/2.0**64.0
1.0/2.0**64.0	2.0**64.0
2.0**64.0	2.0**64.0

Debugging

- **Debugging** เป็นทักษะที่มาจากการเรียนรู้และประสบการณ์
  - ไม่ยากที่จะเรียนรู้
  - และสามารถนำไปประยุกต์ใช้กับ field อื่น ๆ ได้
    - เช่นการวินิจฉัยโรค,
    - การทดลองทางวิทยาศาสตร์
    - หรือแม้กระทั่งการซ่อมเครื่องจักร

Debugging a Function

- หากฟังก์ชันทำงานผิดพลาด มีความเป็นไปได้ที่จะมาจากสาเหตุ 3 ข้อต่อไปนี้
  1. มีความผิดพลาดใน argument ที่ส่งเข้ามาในฟังก์ชัน
  2. มีความผิดพลาดในตัวฟังก์ชันเอง
  3. มีความผิดพลาดในค่า return หรือการนำค่า return ของฟังก์ชันไปใช้



Debugging a Function [2]

1. ความผิดพลาดใน argument ที่ส่งเข้ามาในฟังก์ชัน
   - ตรวจสอบได้โดยการใช้ฟังก์ชัน `print()` เพื่อแสดงค่า argument ที่รับเข้ามา
2. เช่นเดียวกับความผิดพลาดกรณีที่เกี่ยวกับค่า `return`
   - ใช้ฟังก์ชัน `print()` ก่อนที่จะมีการ return ทุกกรณี
   - ถ้าไม่มีอะไรผิดพลาด ให้พิจารณาการเรียกใช้ฟังก์ชันจากฟังก์ชันอื่น และการนำค่า `return` ไปใช้
3. แต่หากฟังก์ชัน `print()` ก่อนการ return มีค่าที่ผิดแสดงว่าปัญหาอยู่ในฟังก์ชันที่เรากำลังพิจารณา
4. อาจใช้เครื่องมือช่วย debug (Debugger) ได้ - แต่หลายคนเชื่อว่า ฟังก์ชัน `print()` เป็นเครื่องมือในการ debug ที่ดีที่สุด

Some Debugging Hints

- **Look for the usual suspects. E.g., have you**
  - ส่ง argument ให้ฟังก์ชันผิดลำดับ
  - สะกดชื่อผิด พิมพ์ชื่อด้วยอักษรตัวเล็กทั้ง ๆ ที่จริง ๆ ต้องเป็นตัวใหญ่
  - ลืม reinitialize ตัวแปรที่ต้องนำมาใช้อีก
  - ใช้เครื่องหมาย เท่ากับ == เปรียบเทียบ float
  - ลืมว่าฟังก์ชัน built-in บางอันเปลี่ยนข้อมูลเริ่มต้นด้วย
  - สับสนการเปรียบเทียบค่า กับการเปรียบเทียบตัว data object
  - และอื่น ๆ (ที่พลาดเป็นปกติ)

Some Debugging Hints [2]

- อย่าลืมว่าบางที bug อาจจะไม่ได้เป็นอย่างที่เราคิด
  - ไม่อย่างนั้นคงหาเจอนานแล้ว
  - วิธีหนึ่งที่ช่วยได้คือการตัดสิ่งที่เป็นไปไม่ได้ออก

> *"Eliminate all other factors, and the one which remains must be the truth."*
> - Sherlock Holmes (The Sign of Four)

- เลิกถามตัวเองว่าทำไมโปรแกรมถึง*ไม่ทำงาน*ในลักษณะที่เราต้องการ
  - ให้เปลี่ยนคำถามเป็น ทำไมโปรแกรมถึง*ทำงานในลักษณะ*ที่เป็น*ตอนนี้* จะทำให้เข้าใจโปรแกรมได้ง่ายขึ้น

Some Debugging Hints [5]

- ลองอธิบายปัญหาให้คนอื่นฟัง ทุกคนมีจุดบอด หรือสิ่งที่มองข้าม
  - บางทีการอธิบายปัญหาให้คนอื่นฟัง จะทำให้เรามองเห็นสิ่งที่มองข้ามไปได้
  - เช่น ลองอธิบายว่า ทำไมถึงแน่ใจว่า บางส่วนของโปรแกรมเป็นส่วนที่ไม่ได้สร้าง bug แน่ ๆ
- อย่าเชื่อทุกอย่างที่อ่าน บางที documentation หรือ comment ที่มากับ code (ของคนอื่น) อาจจะผิด

Some Debugging Hints [6]

- หยุดการ debug ไว้ชั่วคราวแล้วเปลี่ยนไปทำ documentation หรือเขียน comment แทน
  - ช่วยเปลี่ยนมุมมองในการมองปัญหา
- หยุด แล้วกลับมา debug ต่อทีหลัง (walk away, and try again tomorrow)
  - อาจจะเสร็จช้ากว่าเดิม แต่ใช้เวลาหาน้อยลง

When You Have Found "The" Bug

เมื่อหา bug เจอ ควรพิจารณาวางแผนก่อนที่จะรีบแก้ bug ให้หายไป

- จุดมุ่งหมายจริง ๆ ไม่ใช่เพื่อการ แก้ bug เฉพาะตัวนี้ แต่เพื่อเขียนโปรแกรมที่ไม่มี bug
- ถามตัวเองว่า bug ที่เจอเป็นสาเหตุที่เป็นต้นตอจริง ๆ หรือ จริง ๆ แล้วแค่แสดงให้เห็นปัญหาที่สะสมมาจากส่วนอื่น
- พิจารณาผลกระทบที่จะเกิดขึ้นในส่วนอื่น ๆ หลังจากการแก้ bug ด้วย
  - อาจสร้าง bug ตัวใหม่
  - อาจจะทำให้โปรแกรมทำงานซับซ้อน จนทำให้ช้าลง
- อาจจะเป็นโอกาสที่ดีในการแก้ design ในบางส่วนของ code ด้วย

References

- **Introduction to Computation and Programming Using Python, Revised - Guttag, John V.**
- **Think Python: How to Think Like a Computer Scientist**