Python Lecture:
Understanding Errors

Syntax, Semantic, and Logical Errors

Simple Python Programs

- Before talking about errors, I will remind you what a basic Python program looks like.
- We will start with the simplest possible programs.
- Then we will see how they can go wrong.

Example 1: Printing a Message

A very simple Python program:

```
print("Hello, world!")
```

- `print` is a built-in Python command that shows text on the screen.
- `"Hello, world!"` is a piece of text. In Python this is called a **string**.
- The parentheses `( )` tell Python what we want to print.

Example 2: A Tiny Calculation

Python can also do calculations:

```
print(2 + 3)
```

- Here Python adds 2 and 3.
- The result of the addition is then printed on the screen.
- Python understands basic arithmetic like +, -, *, /.

Example 3: Printing Multiple Items

Python can print multiple values in one line:

```
print("Sum:", 10 + 5)
print("My age is", 20)
```

- `print` can accept multiple items separated by commas.
- This avoids many beginner mistakes when mixing text and numbers.

Example 4: Simple Variables

A variable stores a value:

```
x = 10
y = 5
print("x =", x)
print("y =", y)
print("x + y =", x + y)
```

- A variable name is like a label for a value.
- We can use variables inside calculations.

Mini Exercise 1

Try these in Python:

1. Print your own name:

```
print("Your Name Here")
```

2. Print the result of `10 + 20`.
3. Print a short sentence, for example:

```
print("I am learning Python.")
```

While you try these, notice what happens when the code is correct.

Mini Exercise 1 (Extra)

Try these:

1. Create two variables and print their sum:

```
a = 7
b = 8
print("Sum =", a + b)
```

2. Print the result of:

```
print(100 - 35)
print(6 * 7)
print(20 / 4)
```

What Is an Error?

- When I write a program, I am giving exact instructions to the computer.
- If something is wrong with these instructions, we say there is an **error**.
- Errors are completely normal. Every programmer sees errors every day.
- Our goal is not to avoid errors forever, but to understand and fix them.

Error and Bug

- **Error**: a problem in the program or something that goes wrong when the program runs.
- **Bug**: an informal word programmers use for a problem in the program.
- In everyday programming, people use “bug” and “error” quite loosely.
- In this lecture, I will focus on three important kinds of errors in Python.

Three Main Types of Errors

In this lecture I will talk about:

1. **Syntax errors**
2. **Semantic errors**
3. **Logical errors**

- These three types appear again and again in programming.
- Understanding them will make you much faster at fixing problems.

Quick Check: Identify the Type

For each line below, I will decide whether it is: **Syntax**, **Semantic**, or **Logical**.

- `print("Hello)`
- `print(age)` (but age is never defined)
- `print(10 - 3)` (but I wanted addition)

I will revisit this idea again at the end.

Syntax

- Every programming language has rules about how code must be written.
- These rules describe the correct structure of the code:
  - where to put parentheses,
  - how to use quotes,
  - how to separate items, and so on.
- These rules are called the **syntax** of the language.
- Syntax is similar to grammar in a human language.

What Is a Syntax Error?

A **syntax error** occurs when:

- the program text is not valid Python;
- Python cannot even start running the program;
- Python shows a message that includes `SyntaxError`.

Important:

- With a syntax error, the program does **not run at all**.

Syntax Error Example 1: Missing Quote

Incorrect:

```
print("Hello, world!)
```

What is wrong here?

- The string starts with a double quote ", but there is no closing double quote.
- Python expects another " before the closing parenthesis.

Correct:

```
print("Hello, world!")
```

Syntax Error Example 2: Missing Parenthesis

Incorrect:

```
print("Hello, world!"
```

- The opening parenthesis (is present.
- The closing parenthesis) is missing.
- Python reaches the end of the line and finds an open parenthesis.

Correct:

```
print("Hello, world!")
```

Syntax Error Example 3: Extra Symbol

Incorrect:

```
print("Hello") $
```

- The $ symbol at the end does not belong there.
- Python does not expect it and will complain.

Correct:

```
print("Hello")
```

Syntax Error Example 4: Indentation at Start

Suspicious:

```
  print("Hello")
```

- There are spaces before `print`.
- In Python, spaces at the beginning of the line are meaningful.
- At the very top level, starting with spaces can cause indentation problems.
- For now, when we write simple one-line programs, I avoid extra spaces at the start.

Safer:

```
print("Hello")
```

Syntax Error Example 5: Missing Comma in print

Incorrect:

```
print("Sum:" 10 + 5)
```

- In `print`, items must be separated properly (comma or plus with strings).
- Here Python sees two things next to each other without an operator.

Correct:

```
print("Sum:", 10 + 5)
```

Syntax Error Example 6: Invalid Variable Name

Incorrect:

```
2name = "Ali"
print(2name)
```

- Variable names cannot start with a number.
- Python will stop with a syntax error.

Correct:

```
name2 = "Ali"
print(name2)
```

Syntax Error Example 7: Two Operators Together

Incorrect:

```
print(10 + * 2)
```

- Python cannot understand + * together.
- This breaks Python grammar.

Correct:

```
print(10 * 2)
```

How Python Shows Syntax Errors

When a syntax error happens, Python:

- shows the line number where it *noticed* the problem;
- often shows a small arrow ^ under or near the problem;
- prints a message that includes the word `SyntaxError`.

Sometimes:

- the real mistake is slightly before the place where Python reports the error.

Mini Exercise 2: Fix the Syntax

For each example:

- Run it and read the error message.
- Find the syntax error.
- Fix it.

A)

```
print("Python is fun!)
```

B)

```
print("Result is:", 2 + 3
```

Mini Exercise 2 (continued)

C)

```
print("Welcome" "to" "Python"))
```

D)

```
prnt("Hello")
```

For D:

- The syntax is valid, but `prnt` is not a defined name.
- This is very close to a syntax error, but actually it is a different type of error.
- I will come back to this when I talk about semantic errors.

Mini Exercise 2 (Extra Practice)

Fix these syntax errors: E)

```
print("Total:", 10 + 5))
```

F)

```
print("Hello"
print("World")
```

G)

```
name = "Ali
print(name)
```

Mini Exercise 2 (Extra Practice continued)

H)

```
print(8 / / 2)
```

I)

```
print("Value:", 5 5)
```

J)

```
print("Done") %
```

Syntax Errors: Key Ideas

- Syntax errors mean that Python cannot read the program as valid code.
- They often come from:
  - missing quotes,
  - missing parentheses,
  - extra symbols,
  - misplaced punctuation.
- Python tells me:
  - the type of syntax error,
  - and the line where it noticed the problem.

What Does "Semantic" Mean?

- "Semantics" refers to **meaning**.
- A **semantic error** happens when:
  - the code has correct syntax,
  - but the meaning is wrong according to Python's rules.
- Python understands the structure, but cannot actually do what I asked.
- When I run the program, Python stops with an error message.

Semantic Error Example 1: Variable Not Defined Yet

Incorrect:

```
print(number)
number = 10
```

- Python tries to print `number`.
- At that moment, `number` does not exist yet.
- Python shows: `NameError: name 'number' is not defined`.

Correct:

```
number = 10
print(number)
```

Semantic Error Example 2: Mixing Text and Number

Incorrect:

```
age = 20
print("Age: " + age)
```

- `"Age: "` is a string (text).
- age is a number (integer).
- Python does not know how to use + directly between text and number.
- This causes a `TypeError`.

Fixing Example 2

I can fix it in at least two ways: **Option 1: Convert the number to a string**

```
age = 20
print("Age: " + str(age))
```

Option 2: Let `print` handle both values

```
age = 20
print("Age:", age)
```

Semantic Error Example 3: Converting Text to Number

Incorrect:

```
value = int("hello")
print(value)
```

- `int(...)` tries to convert its argument to an integer.
- The text `"hello"` is not a valid number.
- Python shows a `ValueError`.

Correct:

```
value = int("123")
print(value)
```

Semantic Error Example 4: Name Typo

Incorrect:

```
prnt("Hello")
```

- The syntax is valid: this looks like a function call.
- Python understands it as "call something named `prnt`".
- But there is no such name, so Python shows `NameError`.

Correct:

```
print("Hello")
```

Semantic Error Example 5: Division by Zero

Incorrect:

```
print(10 / 0)
```

- The syntax is correct.
- But dividing by zero is not allowed.
- Python stops with `ZeroDivisionError`.

Correct idea:

```
print(10 / 2)
```

Semantic Error Example 6: Wrong Function Argument Type

Incorrect:

```
value = int(12.5, 3)
print(value)
```

- `int(x, base)` expects `x` as a string if base is given.
- Here, the meaning is wrong, so Python raises a `TypeError`.

Correct:

```
value = int("12", 3)
print(value)
```

Semantic Error Example 7: Calling a Method That Doesn't Exist

Incorrect:

```
text = "hello"
text.append("!")
print(text)
```

- Strings do not have an append method.
- Python raises `AttributeError`.

Correct idea:

```
text = "hello"
text2 = text + "!"
print(text2)
```

Mini Exercise 3

Predict, then run these examples. A)

```
x = 5
y = "7"
print(x + y)
```

- Will this run?
- If not, what kind of error is it?

Mini Exercise 3 (continued)

B)

```
a = "100"
b = int(a)
print(b + 50)
```

- Will this run?
- What will it print?

C)

```
print(total)
total = 10 + 20
```

- What error appears here?
- How can we fix it?

Mini Exercise 3 (Extra Practice)

Identify the semantic error and fix it: D)

```
print(5 / 0)
```

E)

```
name = "Ali"
print(name + 10)
```

F)

```
value = int("12.5")
print(value)
```

Mini Exercise 3 (Extra Practice continued)

G)

```
text = "Python"
print(text[10])
```

- Hint: index positions start at 0 and must be within the string length.

H)

```
n = "5"
print(n * "3")
```

Semantic Errors: Key Ideas

- The code has correct syntax, but the meaning is wrong for Python.
- Python often stops while running with an error message.
- Common examples:
  - using variables before giving them a value (`NameError`);
  - mixing types incorrectly, such as text + number (`TypeError`);
  - converting invalid text to a number (`ValueError`).

What Is a Logical Error?

- A **logical error** occurs when:
  - the program runs without syntax or semantic errors,
  - but the result is wrong.
- Python does not show any error message.
- I have to notice the mistake by checking the output.
- The program does exactly what I wrote, but not what I actually wanted.

Logical Error Example 1: Wrong Formula

Incorrect:

```
# Convert Celsius to Fahrenheit (incorrect formula)
celsius = 25
fahrenheit = celsius * 5 / 9 + 32
print("Temperature in Fahrenheit:", fahrenheit)
```

- This program runs without any error messages.
- But the formula is wrong. The correct formula is:

$$F = C \times \frac{9}{5} + 32$$

Correcting Example 1

Correct:

```
celsius = 25
fahrenheit = celsius * 9 / 5 + 32
print("Temperature in Fahrenheit:", fahrenheit)
```

- Now the formula matches what we want.
- The program still runs without errors, but now the result is correct.

Logical Error Example 2: Wrong Average

Incorrect:

```
# We want the average of three numbers.
a = 10
b = 20
c = 30

average = a + b + c / 3
print("Average:", average)
```

- Python uses the normal order of operations.
- It first computes `c / 3`, then adds `a` and `b`.
- The formula does not match the idea "(a + b + c) / 3".

Correcting Example 2

Correct:

```
a = 10
b = 20
c = 30

average = (a + b + c) / 3
print("Average:", average)
```

- The parentheses make sure the sum is computed before the division.
- Now the result matches the intended average.

Logical Error Example 3: Printing the Wrong Variable

Incorrect:

```
# We want to add 2 to x and show the result.
x = 10
y = x + 2
print("Result:", x)
```

- The code runs without errors.
- But the printed value is `x`, which is still `10`.
- The result we wanted is in `y`, not in `x`.

Correcting Example 3

Correct:

```
x = 10
y = x + 2
print("Result:", y)
```

- Now the program prints the updated value.
- The logic of the program matches our intention.

Logical Error Example 4: Wrong Discount Calculation

Incorrect:

```
# 10% discount on 500 should be 450
price = 500
discounted = price - 10
print("Discounted price:", discounted)
```

- The program runs, but subtracting 10 is not a 10% discount.
- The correct discount is 10% of the price.

Correcting Example 4

Correct:

```
price = 500
discounted = price - (price * 10 / 100)
print("Discounted price:", discounted)
```

Logical Error Example 5: Wrong Area of Rectangle

Incorrect:

```
# Area should be length * width
length = 8
width = 3
area = length + width
print("Area:", area)
```

- The program runs, but the formula is incorrect.
- Area of rectangle is multiplication, not addition.

Correcting Example 5

Correct:

```
length = 8
width = 3
area = length * width
print("Area:", area)
```

How I Detect Logical Errors

- **Check by hand**:
  - do the calculation on paper;
  - compare with the program output.
- **Insert print statements**:
  - print intermediate values to see what is happening;
  - check if each step is correct.
- **Try different values**:
  - change the numbers and see if the result still makes sense.

Mini Exercise 4

A)

```
# The total price of 3 items, each costs 50.
price = 50
total = price + 3
print("Total price:", total)
```

- What total do we actually want?
- What does this program print?
- How can we fix it?

Mini Exercise 4 (continued)

B)

```
# We want to convert minutes to hours.
minutes = 120
hours = minutes / 100
print("Hours:", hours)
```

- How many hours are there in 120 minutes?
- Is dividing by 100 correct?
- Which number should we divide by instead?

Mini Exercise 4 (Extra Practice)

Fix the logic (program runs, but result is wrong): C)

```
# 3 items, each costs 80
price = 80
total = price + price + price + price
print("Total:", total)
```

D)

```
# Convert rupees to dollars (example rate 280)
rupees = 5600
dollars = rupees * 280
print("Dollars:", dollars)
```

Mini Exercise 4 (Extra Practice continued)

E)

```
# Average of 4 numbers
a = 5
b = 10
c = 15
d = 20
avg = a + b + c + d / 4
print("Average:", avg)
```

F)

```
# Simple interest: P * R * T / 100
P = 1000
R = 10
T = 2
SI = P + R + T / 100
print("Simple Interest:", SI)
```

Logical Errors: Key Ideas

- The program runs without error messages.
- The result is wrong because the logic is wrong.
- The computer did exactly what the program said, but the program did not match the idea in our mind.
- I must use my own thinking and checks to catch these errors.

Three Types of Errors: Overview

Syntax error

- Program text is not valid Python.
- Python cannot start running the program.
- Example: missing quote, missing parenthesis, extra symbol.

Semantic error

- Program text is valid Python.
- While running, Python finds something it cannot do.
- Example: undefined variable, mixing text and number incorrectly.

Three Types of Errors: Overview (continued)

Logical error

- Program runs completely.
- No error message, but the result is wrong.
- Example: wrong formula, wrong variable in output.

In short:

- Syntax: "Python cannot read this."
- Semantic: "Python can read it but cannot do it correctly."
- Logical: "Python can do it, but it is not what I meant."

Final Example: All Three Types

Syntax error:

```
print("Start of program"    # missing closing )
```

Semantic error:

```
value = "10"
result = value + 5        # mixing string and number
print(result)
```

Logical error:

```
a = 10
b = 20
sum = a - b               # should be a + b
print("Sum:", sum)
```

Final Practice (More Mixed Examples)

Identify the type (Syntax / Semantic / Logical) and fix:

1)

```
print("Hello")
print("World"
```

2)

```
x = 10
print(y)
```

3)

```
price = 200
discount = price - (price * 5 / 100)
print("Discounted:", price)
```

Closing Remarks

- Errors are a natural part of programming.
- The important skill is to:
  - recognize the type of error,
  - read the message (if there is one),
  - and correct the problem step by step.
- In future lectures, we will write more complex programs and these three types of errors will appear again.
- Now you know the basic ideas behind syntax, semantic, and logical errors in Python.