

Lecture 6: Loop Exercises

Guided activity, laboratory practice and independent exercises

Guided activity: Investment Growth Explorer

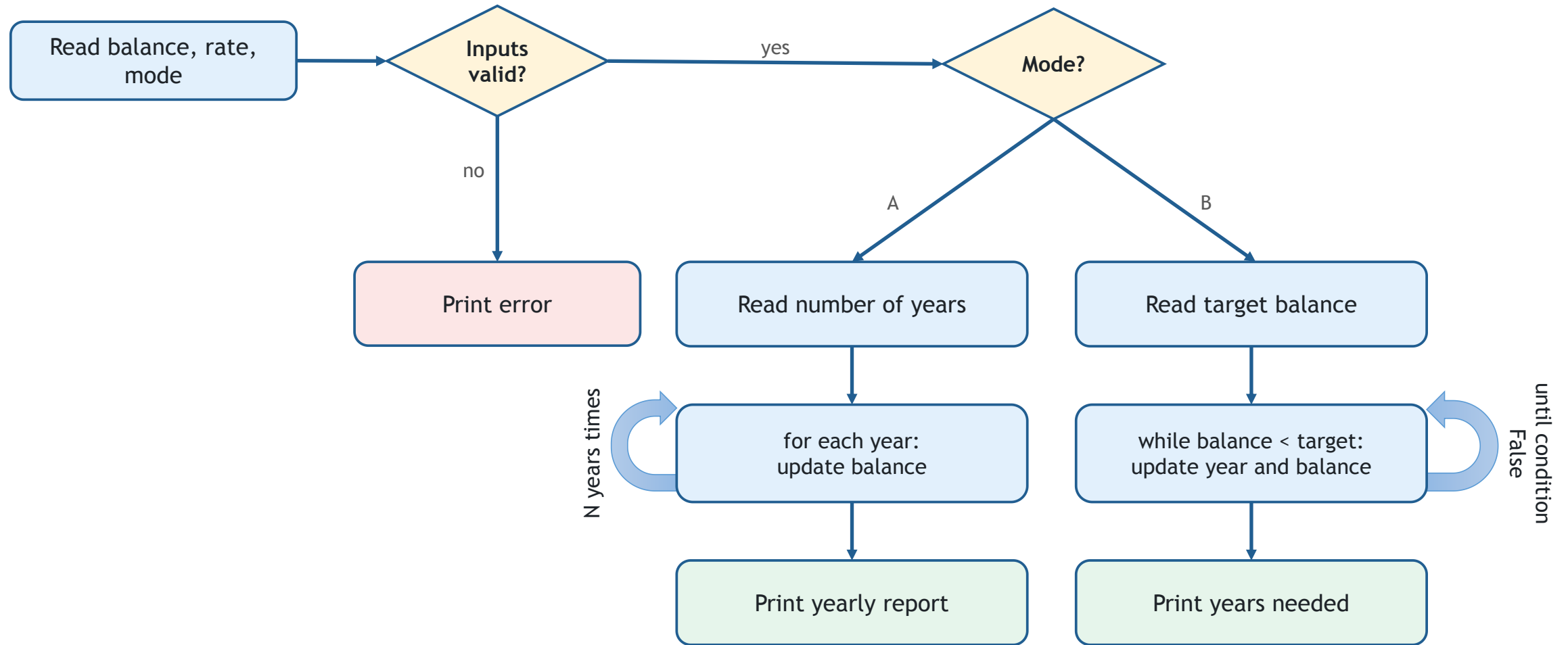
- You have just opened a new bank account; the balance grows by a fixed annual interest rate
- Write a program that helps you forecast your balance
- The program asks for initial balance, annual rate and *one operating mode*:
 - Mode A: asks a number of years and compute the balance after that fixed time
 - Mode B: Ask a target amount and compute how many years are needed to reach the target.

```
interest = balance * rate / 100  
balance = balance + interest
```

```
# equivalent  
balance += balance * rate / 100
```

Known repetitions → for
Unknown repetitions → while

Guided activity: control-flow sketch



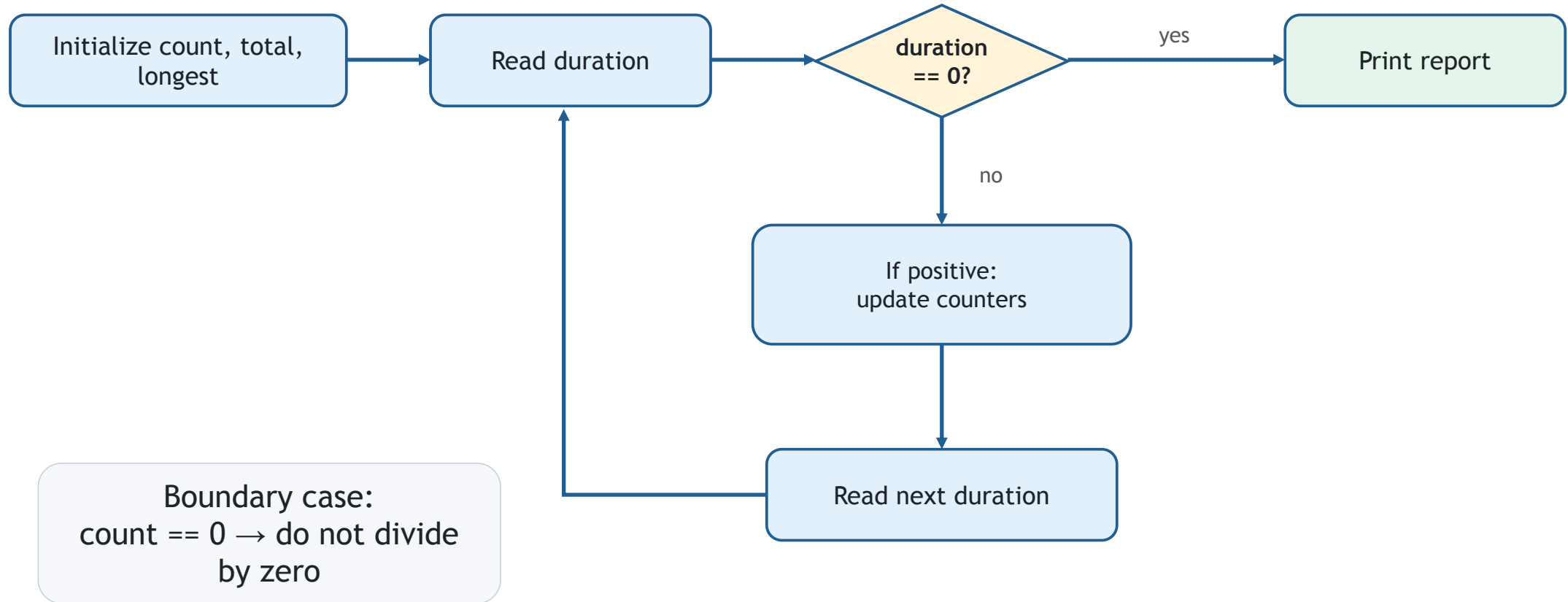
Laboratory 1 : Study Session Statistics

- A student records the duration of study sessions in minutes.
- The number of sessions is unknown in advance.
- Input stops when the **sentinel value** 0 is entered.
- Compute count, total study time, average duration and longest session.

Loop pattern: sentinel-controlled input

```
duration = int(input(...))  
  
while duration != 0:  
    # process duration  
    duration = int(input(...))
```

Laboratory 1 : sentinel loop flowchart



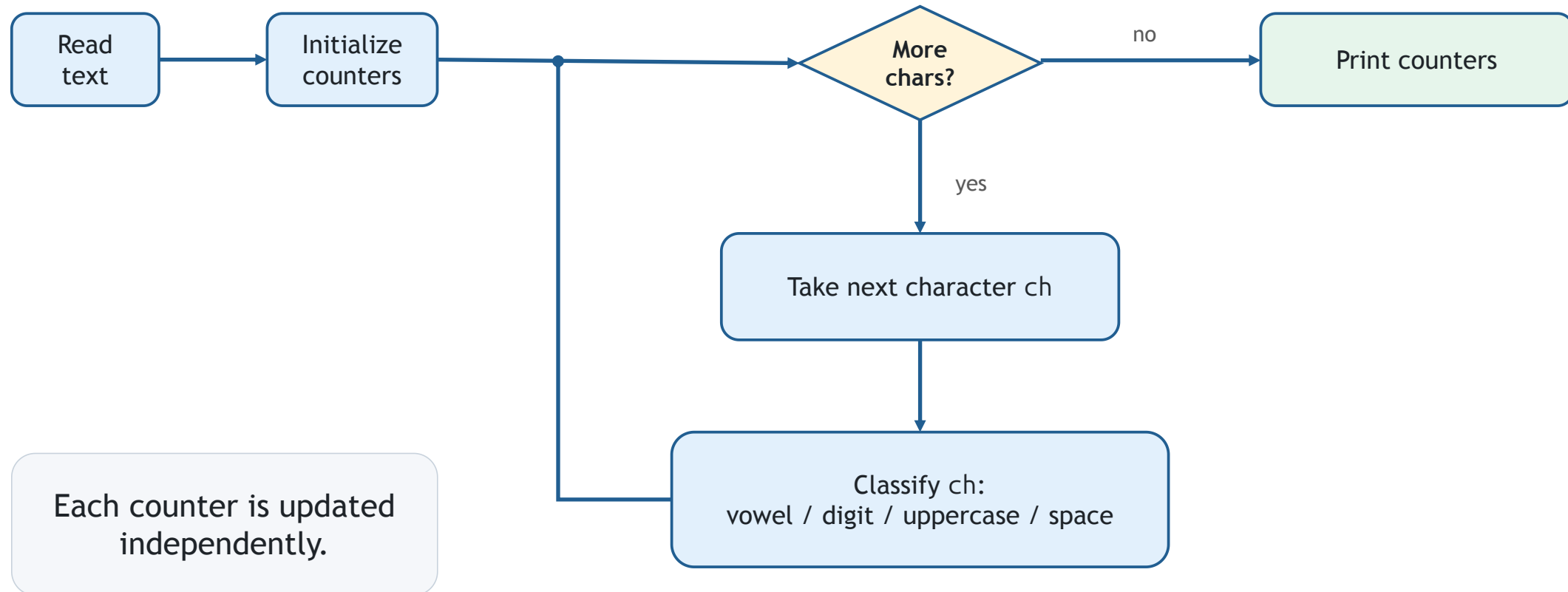
Laboratory 2 : Character Profiler

- Ask the user for one line of text.
- Inspect the string one character at a time.
- Count vowels, digits, uppercase letters and spaces.
- Use a for loop directly over the string.

```
for ch in text:  
    if ...:  
        ...  
    elif ...:  
        ...
```

Useful checks:
`ch.lower()`, `ch.isdigit()`, `ch.isupper()`
`/ .islower()`, `ch == " "`
Remember substring matching using
membership with `in`

Laboratory 2 : character classification flowchart



Independent practice 1: Digit Statistics

- Ask the user for a non-negative integer and analyze its decimal digits *without converting the number to a string*
 - The program must determine:
 - how many digits the number contains;
 - the sum of all its digits;
 - the largest digit.
 - The number should be **processed mathematically**, one digit at a time:
 - The last digit can be obtained using the remainder/module operator %, while integer division // can be used to remove it
- ```
digit = n % 10
n = n // 10
```
- Continue until all digits have been processed. Remember that the value 0 itself contains one digit.

# break: Exit a Loop Early

- keyword `break` immediately stops the nearest enclosing loop, even if the loop condition would normally allow more iterations; the program continues in the upper level
- With nested loops, `break` stops only the innermost loop that contains it.

Use `break` carefully and do not overuse it:

- useful when a loop must stop immediately after a condition is met

However:

- too many `break` → control flow harder to follow;
- may be symptom that the loop condition itself could be clearer;

Rule of thumb: use `break` when it makes the program clearer, not simply to avoid designing a proper loop condition.

```
for row in range(3):
 for col in range(5):
 if col == 2:
 break

 print(row, col)
```

```
Output:
0 0
0 1
1 0
1 1
2 0
2 1
```

The `break` stops the inner `for col` loop (even before the `print()`), but the outer `for row` loop continues with the next row.

# Independent practice 2: PIN Attempts

- A system is protected by the PIN 4827 and allows the user a maximum of three attempts.
- At each attempt, ask the user to enter the PIN.
- If the entered PIN is **correct**, print: “*Access granted*” and immediately stop asking for further attempts.
- If the PIN is **incorrect**, report **how many attempts remain**.
- If all three attempts are used without entering the correct PIN, print: “*Card blocked*”
- The solution must use a `for` loop with `range()`, a Boolean variable to remember whether access was granted, and `break` to terminate the loop early when the correct PIN is entered.

# Independent practice 3: Dice Simulation

- Use Python's `random` module to simulate repeatedly rolling a six-sided die.
- Ask the user how many times the die should be rolled. The number of rolls must be positive.
- During the simulation, compute:
  - how many times the value 6 appears;
  - the observed frequency of sixes;
  - the average value of all generated rolls.
- Generate each result using: `random.randint(1, 6)` (endpoints included)
- Do not store all the generated values; instead, update counters and accumulators during the loop.
- For a sufficiently large number of rolls, the observed frequency of sixes should tend toward approximately:  $1 / 6 \approx 0.167$

# Independent practice 4: Power Table with Nested Loops

- Create a table containing powers of several integer bases.
- Ask the user for:
  - the largest base, `max_base`;
  - the largest exponent, `max_power`.
- For every base from 1 to `max_base`, print its powers from exponent 1 to `max_power`.
- For example, if:
  - `max_base = 3`
  - `max_power = 4`
- the output should be:

|   |   |    |    |
|---|---|----|----|
| 1 | 1 | 1  | 1  |
| 2 | 4 | 8  | 16 |
| 3 | 9 | 27 | 81 |
- Use **two nested for loops**: the outer loop controls the base and the inner loop controls the exponent. Values belonging to the same base should be printed on the same row.

# Loop-design self-check

- What exactly is repeated?
- Is the number of repetitions known? → consider for
- If not known, what condition keeps the loop running? → consider while
- Which variables must be initialized **before** the loop?
- Which variables are updated at each iteration?
- Which boundary case could reveal an off-by-one or infinite-loop error?