#17_Preventing_Errors

30/05/2025

- Two main approaches to error prevention
  1. LOOK BEFORE YOU LEAP (LBYL)
  2. IT'S EASIER TO ASK FOR FORGIVENESS THAN PERMISSION (EAFP)

LBYL: LOOK BEFORE YOU LEAP

- Check for potential errors before executing code that might fail.

```
1  def lower_first(word):
2      # Ensure word is a string
3      if type(word) != str:
4          return word
5
6      # Ensure word contains at least one character
7      if len(word) == 0:
8          return word
9
10     # We now know that word is a string that contains at
11     # least one character. That means the following code will
12     # run without generating an error.
13     return word[0].lower() + word[1:]
14
15 print(lower_first("FOO"))  # Output: "fOO"
16 print(lower_first(32))     # Output: 32
```

guard clause: returns word 'as is' if not a string

guard clause: returns word as 'is'. i.e., empty

Guard Clauses

- LBYL uses one or more guard clauses to ensure data meets the specific preconditions a function expects.
- Two guard clauses above on lines 3-4 and 7-8.

30/05/2025

When to use Guard Clauses

- Best used when a function cannot assume that its arguments are valid
- Invalid arguments can have incorrect:
  - types
  - structures
  - values
  - properties
- No need to include guard clauses if you have trust in your program or section of program will always be valid. Consider the example earlier, if you trust there will never be an empty string, you might not need the guard clause on lines 7 & 8.

EAFP: IT'S EASIER TO ASK FORGIVENESS THAN PERMISSION

- This approach involves trying an operation and handling any exceptions that arise.
- EAFP approach assumes the code will execute successfully
- Handles exceptions only if something goes wrong.

```
def lower_first(word):
    try:
        return word[0].lower() + word[1:]
    except (TypeError, IndexError):
        return word # Handle exceptions by returning 'word' as is

print(lower_first("FOO")) # Output: "fOO"
print(lower_first(32))    # Output: "32"
```

30/05/2025

- Exception handling often occurs in EAFP code due to its nature of trying operations without explicit checks. This was covered off in #16_Errors

- EAFP generally preferred in the Python Community.

Detecting Edge Cases

- Edge cases are the instances of challenging underlying assumptions in your code.

Step 1:

- Analyse the inputs to your code.
- Generally its the arguments in functions that can lead to unexpected behaviour.

examples:

- if an argument should be numeric, will the code still function if the argument is zero or negative?
- passing a float when code expects an integer
- in lower_first function example earlier, the empty ("") string is an edge case.
- other instances may be cases such as when strings have leading/trailing spaces or contain special characters, or only spaces.

Step2:

- Consider contemplating how specific combinations of values can lead to unforeseen conditions.

Planning Your Code

- Write out the common use cases of a new function and check how the function handles them.

30/05/2025

```
countries = ['Australia', 'Cuba', 'Senegal']

alpha_insert(countries, 'Brazil') #Inserts 'Brazil' into countries

print(', '.join(countries)) #Outputs "Australia, Brazil, Cuba,
                                      Senegal"
```

- Some use cases for the above function
- Want to make sure alpha_insert can handle these use cases.

```
alpha_insert([], 'Brazil') #Inserting into an empty list
alpha_insert(['Brazil'], 'Australia') # At the beginning of a list
alpha_insert(['Brazil'], 'Cuba')   # At the end of a list
alpha_insert(['Brazil'], 'Brazil')  # Duplicate entry
```

- Focus on the basic use cases to begin with.
  - check
  - revise list of use cases if a specific case fails.

- Difficult to maintain code that checks argument types and is unneeded.
  - For example, passing a number when a function is expecting a string.