Software Development Lifecycle

Developing software is a challenging task, especially when needing to ensure that software meets the needs of users, is free of major bugs, and can be extended and maintained by future developers. Let's look at a workflow that many developers use to structure their development process.

1. Problem/Idea

Most program development starts with a problem to solve or idea to help the intended users.

2. Requirements

Carefully consider what your program will need to do, and how it will need to do it (e.g. How will users access it?).

3. Design

Your design will acts as a plan for developing the program. You will often update your design throughout development.

4. Source Code

We write source code in a programming language (e.g. Scratch, Python) to construct a program that the computer can execute.

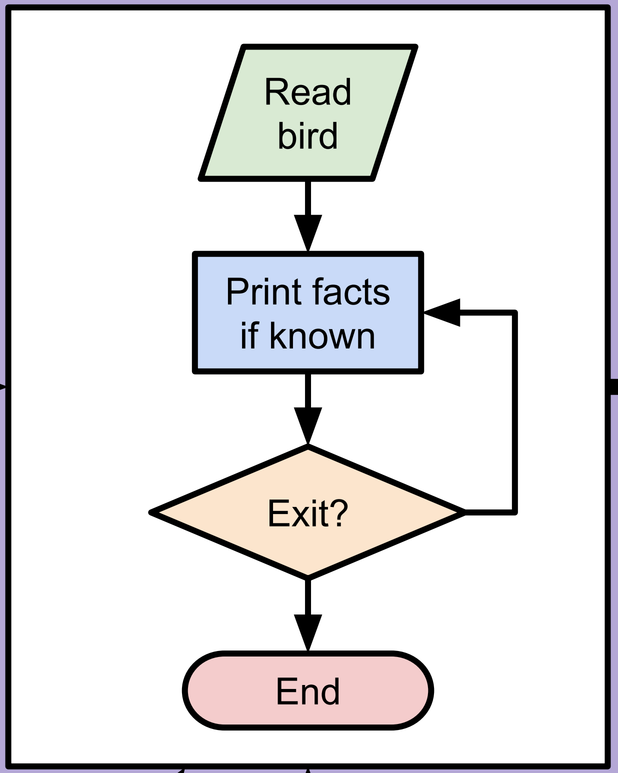
5. Testing

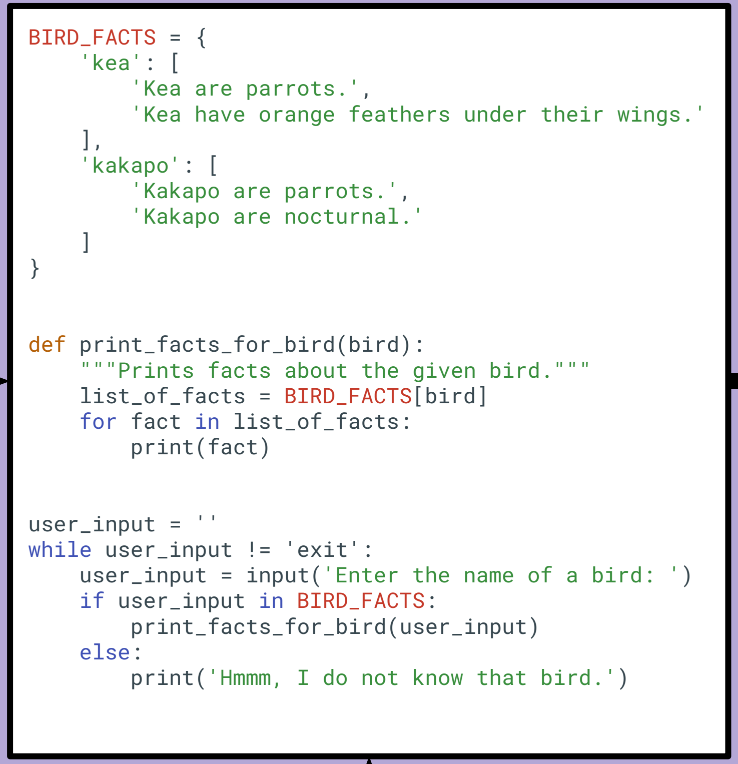
When testing, consider:

- **Verification**: Are the requirements met?
- **Validation**: Does it meet our users' needs?

Birds want to find out facts about other birds!

- How will users interact with the program?
- Which birds should be included?
- What kinds of facts should we include?





```
BIRD_FACTS = {
    'kea': [
        'Kea are parrots.',
        'Kea have orange feathers under their wings.'
    ],
    'kakapo': [
        'Kakapo are parrots.',
        'Kakapo are nocturnal.'
    ]
}


def print_facts_for_bird(bird):
    """Prints facts about the given bird."""
    list_of_facts = BIRD_FACTS[bird]
    for fact in list_of_facts:
        print(fact)


user_input = ''
while user_input != 'exit':
    user_input = input('Enter the name of a bird: ')
    if user_input in BIRD_FACTS:
        print_facts_for_bird(user_input)
    else:
        print('Hmmm, I do not know that bird.')
```

- Are the correct facts printed for each bird?
- Can the user exit the program?
- What happens if the user mis-types a bird name?

Algorithm Design

Designing an algorithm for a program often involves the following activities:

Decomposition

Breaking down a large problem into smaller tasks.

Sequencing

Arranging the steps to complete a task in the correct order.

Abstraction

Recognising common patterns between problems that can be solved generically using techniques like *iteration* and *functions*.

Algorithm

An algorithm describes the steps your program will perform to complete its task. Flow charts are helpful tools for documenting algorithms.

6. Debugging

Testing often reveals problems ("bugs") with your software. Debugging is the process of finding the source of these bugs and fixing them.

8. Feedback

You should carefully consider feedback from your users, which will often lead to new requirements.

7. Deploying to Users

You should aim to get your program to users as soon as possible to get feedback to inform development.

9. Documentation!

Throughout development, you should document your work to help future developers (including yourself!). You can write specifications, design diagrams, code comments, and user guides.