Solving problems

Learning Objectives

After completing this unit, you'll be able to:

- distinguish various steps in the problem solving process
- Google search for solutions more effectively

Quote

"Everyone in this country should learn to program a computer because it teaches you to think."- Steve Jobs.

(As a tekkie you may have heard this quote before)

This 'thinking' Steve Jobs refers to, what does it mean?
From an IT point of view this 'thinking' (or your cognitive ability) comes, simply put, down to problem-solving skills. Along with other important 'meta-skills' like perspective-taking, story-telling, and connecting the dots, mastering this problem-solving skill is considered essential in our technologically complex world. In fact, many employers value problem-solving more than other skills when looking for new hires. Problem-solving is regarded as the core of any developer or engineering role.

"How do you work through a problem that you haven't encountered before?"

This is the main question that runs throughout your education and also in your further career and personal life. Inspired by the Clever Programmer we will go through the steps you can follow to solve problems effectively.
Of course, the use of Google search, to look up for instance the documentation for an API or framework, is a smart move.
Getting better at *Googling* is almost mandatory for a successful IT career, but actually, a Google search is your last step in the problem-solving process. Before we get into tips on how to (Google) search effectively, let's go through the steps of problem-solving.

Problem solving, a framework

#1. Understand the context/the problem

Know exactly what is being asked. Most difficult problems are difficult because you don't understand them. Most of the time it is necessary to step back and try to understand the context in which you are working or learning. Sometimes knowing where to start is the first step. Facing complex IT problems can make us nervous, the unfamiliarity of the terrain we're in leaves us helpless. When you have no clue about what you're doing, a reasonable thing to do is to hit the pause button and figure out how to proceed.

Just to be clear, fixing IT problems is not the same as being chased by wild animals, so there's absolutely no need to fight, flight or freeze. It all comes down to take the time to think and ask yourself the right questions. Asking the right questions and validating your answers will help you to gather information about the context and identify and understand the problem.

NOTE:When you don't understand the context you are in, for example, because of a new concept you don't yet understand, you can avoid getting stuck by changing your perspective.Suppose you have done your best to understand this new concept, but you have not succeeded yet, it is not because you do not have the ability to understand it, but because you have not yet found the explanation by which you understand this concept. So extend your information search (in various media by various experts).

How do you get started when, for instance, you are expected to develop a TODO app? An app that the user can use to define their tasks. To set priority of the tasks, and of course mark the tasks when it's done. When you first try to code a TODO app yourself you have probably no clue where to start. Based on this context you can ask these questions to yourself to have a better understanding of how to chunk the main task into smaller problems. The trick is to be as concrete as possible.

What is expected of me or what am I building?

Bad answer:
Building an app

Good answer:
Building a TODO app (this is the big problem you want to solve)

When it's clear what you are trying to achieve, then you could dive one level deeper to have a better understanding of how this TODO app works. The follow up question could be:

What kind of functionality does this TODO app have or what do I want the app to do?

Bad answer:
The app can define, prioritise and mark tasks. Also an overview of the tasks

Good answer:
- *The app should be able to give the user an overview of tasks.*
- *The app should be able to add tasks.*
- *The app should be able to prioritise tasks.*
- *The app should be able to mark tasks.*

With one question you could have multiple answers to it. That's fine, these are smaller problems that you actually want to solve. Each of the answers you gave can go one level deeper again to gain exact information about what you need to build. E.g The follow up question to "*The app should be able to give the user an overview of tasks"* can be:

What do I need to build an overview of the tasks?

Bad answer:
I need to show an overview of the tasks on screen.

Good answer:
- I need a list of existing tasks.
- I need to show the existing tasks as a list on screen.

Same again, having multiple answers to a single question is fine. Taking the first answer we can go one level deeper again. E.g The follow up question to "I need a list of existing tasks.*"* can be:

How to build existing tasks?

Bad answer:
Write tasks in code.

Good answer:
- Write a few example of texts

- Use those examples of texts as a list of tasks.

Now, this should be pretty clear how to build "*an overview of the tasks*". When you have done this you are 50% done with the functionality. Repeat the same steps with the second answer **"*I need to show the existing tasks as a list on screen.*"** by asking yourself what the follow up questions are until you exactly know how to build it. It should create focus on your work and clarify your ignorance.

How to know when you understand a problem? **When you can explain it in plain English!** Do you remember being stuck on a problem, you start explaining it, and you instantly see holes in the logic you didn't see before? This is why you should write down your problem, draw a diagram, or tell someone else about it.

#2. Divide

Don't dive into the problem and try to solve it. You will probably get lost along the way. Divide the problem into manageable chunks.
Taking the example of the TODO app. Asking the right question and giving the right answer is the key to chunk down the problems. Think of the functionality as a standalone functionality. Having an overview of the tasks is not dependent on the functionality of adding tasks. That you can prioritise tasks doesn't mean that you need the functionality to mark the tasks to be able to create that. Avoid using words like AND or OR in your answer. It narrows down to one specific thing that you can focus on.

#3. Plan

Then, solve each sub-problem one by one. Begin with the simplest. Simplest means you know the answer (or are closer to that answer). After that, simplest means this sub-problem being solved doesn't depend on others being solved. Once you have solved every sub-problem, connect the dots. Connecting all your *"sub-solutions"* will give you the solution to the original problem.

#4. Stuck?

Writing about a framework to solve problems is one thing...but stuck with a seemingly unsolvable sub-problem is a whole other ballgame. So first off, take a deep breath and remember that bugs or error messages are clues to better understand the problem. Be curious about where you went wrong.

If necessary, reassess, take a step back. Look at the problem from another perspective. Is there anything that can be abstracted to a more general approach? Or reassess a previous step you took. What would happen if you would choose an alternative solution? Sometimes when we get stuck in our own approach a solution could be to start over. Perhaps not from scratch but at least a few steps back to try out a different approach.

And now we have arrived at the most commonly used approach to arrive at solutions: the Google search. No matter what your problem is, someone probably already solved it. Even when you already solved the problem it is advisable to look up solutions from others. You can learn a lot from how others solve problems. A pitfall of Google search for solutions is applying found solutions to the main problem. Only look for solutions to sub-problems. Why? Because unless you struggle (even a little bit), you won't learn anything.

#5. Review your work (and the work of others)

Solving problems with a framework like this will help you enhance your productivity. One way to further develop your problem-solving skills is to review the work of your colleagues or colleagues. When you look into the work of others you simultaneously compare it to your own product and approach. In this way, your review of the work of peers will provide you with information to improve your own approach or product.

To improve the outcome of your review you could divide it into 3 tasks:

1. compare your work and formulate the biggest differences between your work and the work of your peer

2. formulate what you have learned from these differences

3. write a judgment on which work is best and why

Takeaways

1. Don't start by trying to solve the problem. First, aim to understand the root of the problem.

2. Use questions to understand the context and problem. rephrase the problem in plain language.

3. Chunk it up. Divide the problem into manageable sub-problems that can be tackled individually.

4. When stuck, rethink the approach, try alternative solutions and if necessary start over.

Google search more effectively

Numerous articles, blogs, websites provide tricks and tips on how to improve your Google search skill. It is impossible to provide a complete overview of these recommendations. Below an overview of general tips, and a list of practical tricks.

#**1 — Keywords**
If you're debugging an error message, pull out any individual information, such as file paths. This allows you to search a wider range of results.
#2 — Source
Where is your error coming from? Is it a library? If so, look up the library's GitHub page (if it has one), and check out the issues page.
#3 — StackOverflow
Is the most popular Q&A website which is used by developers and programmers for many developments and programming purposes
#4 — Describe the issue
If the problem is due to certain circumstances, try and pick out the keywords from this setup in your search term, as in step 1.
#5 — Ask your colleagues!
Also, there are many Slack groups set up that specialise in certain languages, with often fantastic communities who are incredibly helpful.
#6 — Explain it to yourself
Self-explaining is a proven method for better understanding. Try to explain the problem in plain language.

A few search practical tricks

Using the asterisk *
Can't think of all the words? Include an * to tell Google to fill in the blanks.

Using a Specific Domain
If you are searching for a specific code example, it's likely that you'll want to search on StackOverflow. Your search would be as follows:
site: stackoverflow.com could not cast value type *to*

Use “Solved” as an Extra Search Term
Although it’s not an officially supported feature in Google, it does boost your developer productivity by trying to filter out unsolved questions.

Use define:
Easily find the definition of a word without having to go to a dictionary website by using *define:* before your word. Google will deliver the definition and an audio player that offers the word's phonetic pronunciation

https://twitter.com/nickbulljs/status/1355789724808601602