**Introduction**

Imagine you've just finished your program and are ready to share it with users and fellow programmers. While working on it, you probably mostly focused on writing code, so a machine would understand what needs to be done and the final program would be executed correctly. However, how will other people know the way your program works, what it can and cannot do, what tasks it's supposed to solve? Most importantly, will you yourself be able to answer all these questions after some time passes? One way to make sure you and others can make use of the program regardless of time and place is to document your code. Let's find out what ways of documenting code are there.

**Strategies for documenting code**

***Documentation***is any kind of written or illustrated information that accompanies the source code or is embedded into it and aims at explaining how the code works or how to use it depending on whom it's written for (users or developers).

Let's look at some ways to document your code:

* writing explanatory comments (and other types of annotation provided by a particular language) to help understand what a particular piece of code is for (e.g. "This function takes the name of a book, its author and prints a list of stores in the area close to you where you can buy it right now".);
* providing images, block schemes, diagrams, and other illustrative data that graphically shows what's going on (e.g. a diagram that shows the implemented neural network architecture);
* writing README file containing all the necessary information about code utilization (e.g. [README](https://github.com/JetBrains/kotlin/blob/master/ReadMe.md)to Kotlin);
* providing additional information in the form of PDF or HTML files, guides, handbooks, etc.
* creating a wiki for managing your documentation (e. g. wiki for [Python](https://wiki.python.org/moin/Documentation));
* writing release notes/changelogs (e.g. [Slack release notes](https://slack.com/intl/en-ru/release-notes/windows))

To get started and get more information on this, check this [beginner's guide](https://www.writethedocs.org/guide/writing/beginners-guide-to-docs/) to writing documentation from, as stated, "a global community of people who care about documentation".

Now let's take a closer look at some of the strategies listed above.

**README**

**README** is usually a plain text file (if it's called README.md though, it's using markdown markup like the provided example) which serves as an introduction to a project explaining what it does and why. There is no strict content policy for a README file, which, to some extent, will also depend on whom you're writing it for (users or developers). However, you may notice some commonly-used sections like:

* Name
* Description
* Installation
* Usage
* FAQ
* Support
* Contributing
* Contact information
* License
* Acknowledgment

Section names are quite self-explanatory, but you can read [more instructions](https://www.makeareadme.com/) on how to write README files. When the file is ready, put it in the same directory as your project.

**Wiki**

**Wiki** is basically a website that can be edited by multiple users, so it's especially convenient for team projects. Writing a whole wiki is not gonna be easy, so you'll need some tools and recommendations on how to do it. One of the most popular software used for this type of work is [MediaWiki](https://www.mediawiki.org/wiki/MediaWiki" \t "_blank). It's a free and open documentation platform that helps you organize the information about your project in a customizable way. Obviously, this format of documentation gives you much more freedom in design and functionality than a simple README file, so you can create a complete and clickable table of contents, add media files, embed code snippets with correct indentation, etc.

**Release notes/changelogs**

**Release notes** and **changelogs** are specific types of documentation used for informing and keeping track of updates, so it's particularly useful for version control. When something's been changed in the project, you write a release note to inform users and colleagues how exactly this new version differs from the previous one: what's new, what's been fixed, what's deprecated, etc. By writing release notes you will always know at what moment something went wrong or broke, and ask for feedback. Even if you're not planning on publishing these files, it's good practice just to keep these notes for yourself.

**Auto-generated documentation**

There are also external tools that have different functionality that at their core are aimed at helping you with generating documentation for your code. It's important to notice that, for the most part, these tools do not automatically write documentation for you. Since their work is mostly based on the comments embedded in the source code in the first place, they simply convert what's already been written into a more structured and readable form. For example, [Doxygen](https://www.doxygen.nl/index.html" \t "_blank) supports many languages and can generate HTML files from source code. [Sphinx](https://www.sphinx-doc.org/en/master/) was originally developed for Python documentation and is ready to help you with building tables, defining document trees, etc. [Dr. Explain](https://www.drexplain.com/) is a tool for frontend developers, it analyzes the user interface of an application and generates documentation in a fast and user-friendly manner. This list is by no means complete, so you can find something for yourself.

**Pros and cons**

Documenting your code one way or another is quite an important skill to master on your journey to becoming a good programmer (which is our goal!). So let's one more time summarize why it's important and what pitfalls might be there:

Pros:

* helps to refresh your memory
* guides you through the code
* explains how the code works and how to use it
* helps to fix bugs
* allows for version control

Cons:

* takes a lot of time
* takes practice to make it structured and clear
* easily becomes outdated, so needs to be updated all the time
* it's a common practice to write documentation in English which might be a foreign language for you
* constant feedback is needed to make it genuinely good

**Summary**

In this topic, we've learned about software documentation, its types, ways to write it, and its pros and cons. Now we know that this important information helps us use different types of software, and understand/explain how it works. We also looked at some helpful documentation generators. All in all, it's a very powerful tool everyone benefits from. However, it may also take some time and effort to be able to use it proficiently.

What information about a function in the code should you reflect in the documentation? Choose a few correct answers.

**Correct answers:**

* **what arguments the function takes as an input**
* **what the function returns as an output**

Imagine that you and your team are working on a large project and want to write the documentation for it together. Moreover, you know you may need to edit it several times till everybody is satisfied with the result. Which method may be the most convenient for you?

use Wiki

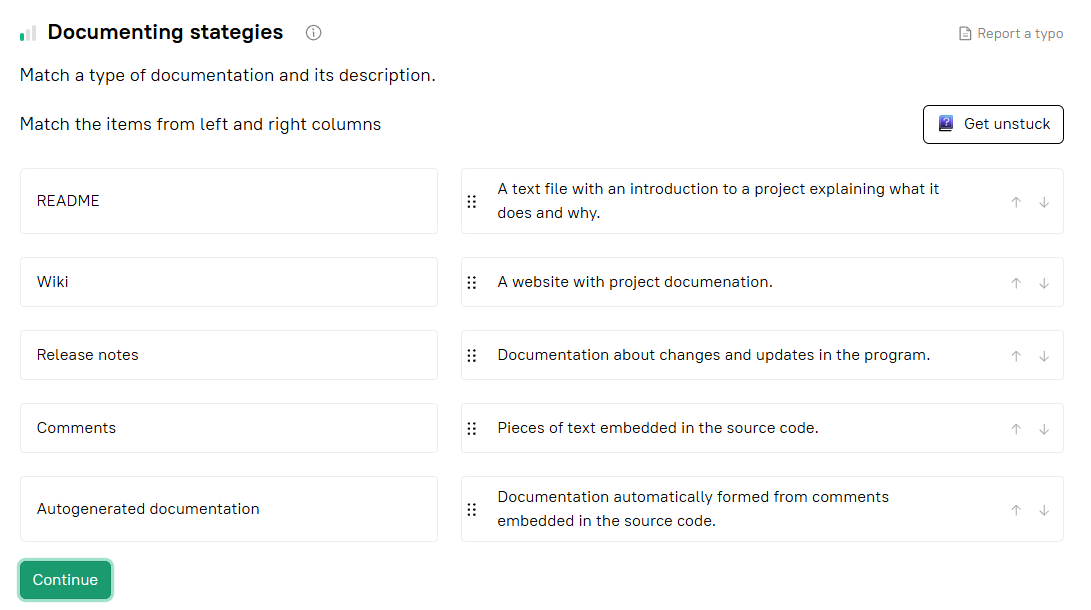
Choose the strategies you might use for documenting your code.

✅ **Correct answers:**

* **provide a PDF file with instructions**
* **write README**
* **create a wiki**
* **write comments**

### ❌ Incorrect:

* **print code**  
  → Not a documentation method; it's static and non-collaborative.
* **get a license**  
  → Important for legal usage, but **not related to code documentation**.



Your company has decided to use some tools to automatically generate documentation for its projects and assigned this task to you. What should you take into account before choosing a particular program?

✅ **Correct answers:**

* **Does it use common formats like HTML and PDF for the output?**
* **Does it convert formatted comments into a more structured and readable form?**
* **Does it automatically generate documentation?**