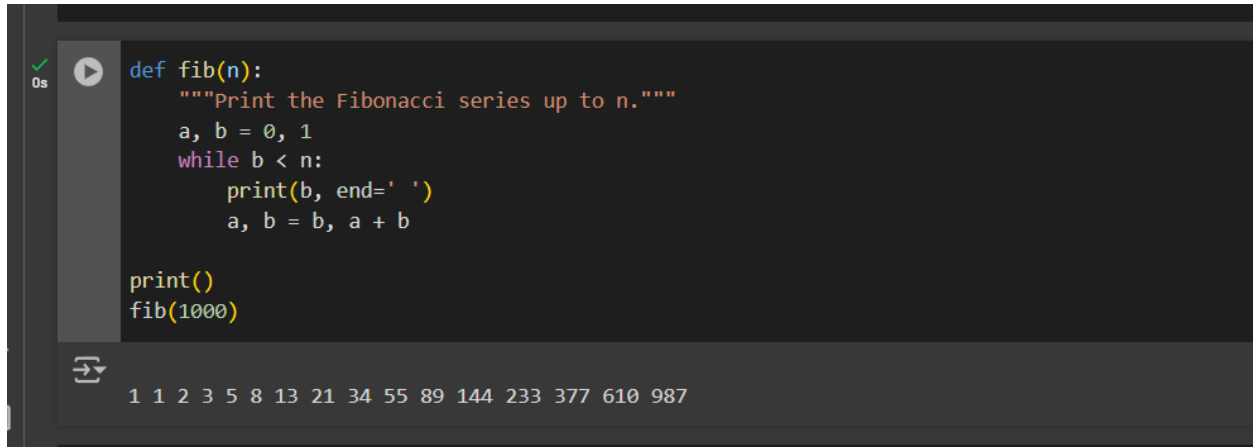


## Code in cython



```
def fib(n):  
    """Print the Fibonacci series up to n."""  
    a, b = 0, 1  
    while b < n:  
        print(b, end=' ')  
        a, b = b, a + b  
  
    print()  
    fib(1000)
```

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

<https://colab.research.google.com/drive/1jcgsvwkdMTzuW7goGWvHnmmEwUvVhwcw->

## Docker

is a cloud-native platform that lets you build, run and share containerized applications through dockerization — the process of creating Dockerfiles that can package all the software needed to run an application into a portable and executable Docker image

## Cloud containerization

Cloud containers act as a virtual homebase for software. These containers run on top of an operating system (OS) and on the back of a comprehensive setup that includes defining the OS and the number of containers needed. They abstract the underlying OS away and allow you to focus solely on the application. This is what makes containers so attractive.

## Containerization use cases

- Potential use cases for cloud containerization include:

- The need for rapid deployment
- Desire for application portability, such as moving applications from one environment to another environment that uses the same OS
- Enterprises that require scalability of containers
- Goal of creating standardization during the development process

## **Docker architecture**

Docker uses a fairly standard client-server model with multiple layers. This infrastructure is a great way to implement continuous integration and development of CI/CD methodologies while providing easy mobility across dev environments — as long as the operating system is the same.

## **Docker client**

This is how users interact with Docker, using the provided command line interface (CLI) to build, run and stop applications. The client may share a host with the background program or connect via a remote host.

## **Docker host**

This environment encompasses the background program, images, containers, network and storage needed to set up to execute and run applications.

## **Docker images**

Docker images are templates (written in YAML — Yet Another Markup Language) that house the instructions for the associated container. These images are built in read-only layers, with build tags used to define which file to reference when the image is used to run the application from the container.

## **Dockerfile**

Dockerfiles contain all the commands and dependencies applications need to run while also helping to create a static image the container uses to run the application. Later on, we explore the anatomy of a Dockerfile.

## **Docker registries**

Users can leverage Docker registries to store, manage and distribute images built from Dockerfiles via commands like `docker push`, `docker pull` and `docker run`. Docker Hub, a public registry, has free and paid plans, but private and third-party registries also exist.

## **Docker containers**

The Docker container runs the images and applications, connecting using restful APIs. The image and file deliver the application to the host server.

## **How to dockerize Python applications**

Developers interested in using Python to build scalable applications may find that Docker makes deployment faster, easier and more secure. Here's how to get started, including step-by-step instructions.

<https://medium.com/capital-one-tech/dockerization-a-practical-guide-to-docker-containerization-6597eecfa0ed>