**A Web App for Agricultural Purpose to detect Plant Leaf Type using Flask**

**ABSTRACT**

Plant classification and species identification are essential tasks in botany, agriculture, and ecological studies. This paper introduces a Flask-based web application designed to harness the power of deep learning for efficient and accessible plant leaf type detection, while also providing comprehensive details about the identified plant, all while abstracting away the complexities of model training and algorithms.

The Flask-Based Plant Leaf Type Detection and Information WebApp (PLTDA) offers an intuitive web interface, allowing users to upload images of plant leaves for analysis. Behind the scenes, PLTDA employs advanced deep learning models to process these images and accurately identify the plant leaf type. Additionally, PLTDA retrieves and displays detailed information about the recognized plant species, including botanical names, characteristics, and growth habits.

PLTDA simplifies the process of plant leaf type identification and botanical information retrieval, making it accessible to a wide range of users, including botanists, horticulturists, and nature enthusiasts. By providing accurate plant recognition and educational details, this application supports plant research, conservation efforts, and gardening practices.

**EXISTING SYSTEM**

There are certain systems available to check plant Type using Plant Leaf. Although an Easy to Use We Interface which can be used by anyone is missing.

**PROPOSED SYSTEM**

The Flask-Based Plant Leaf Type Detection and Information WebApp described in this paper offers an accessible and efficient solution for plant leaf type identification and information retrieval. By integrating deep learning capabilities into a user-friendly interface, PLTDA empowers users to enhance their botanical knowledge, conservation efforts, and gardening experiences, ultimately contributing to a deeper understanding of the plant world.

**HARDWARE REQUIREMENT**

Processor: i3 7th Gen or better

RAM: 8GB or better

Storage: 120GB or more

**SOFTWARE REQUIREMENT**

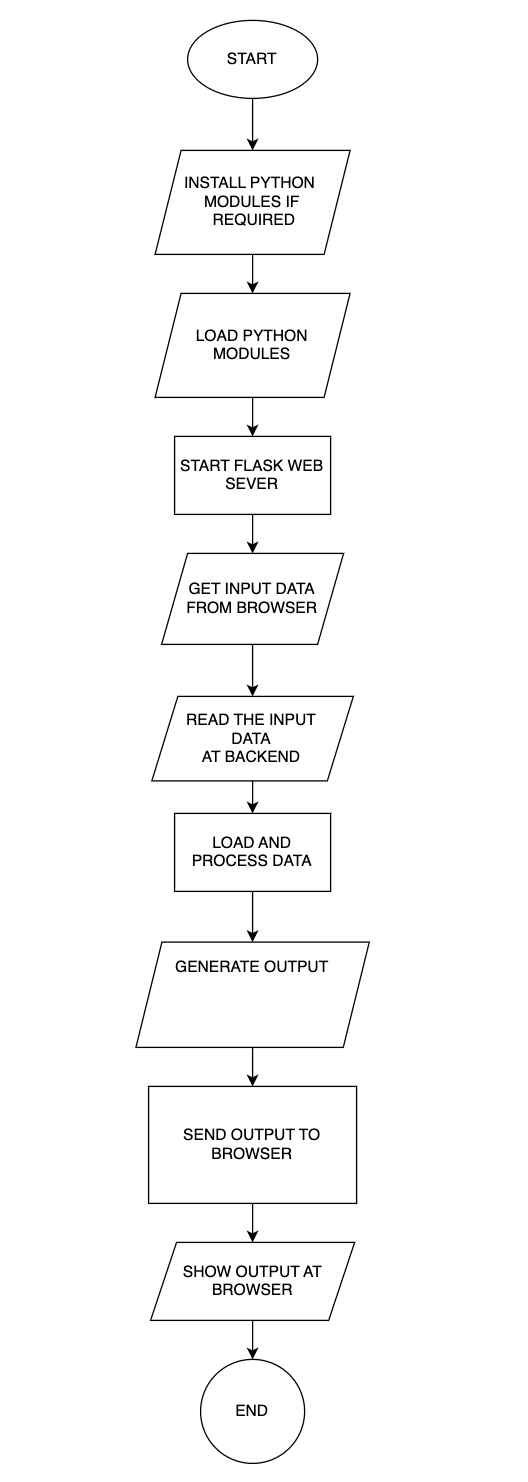
Windows 10

Python 3.9 or newer

Flask  
Web Browser

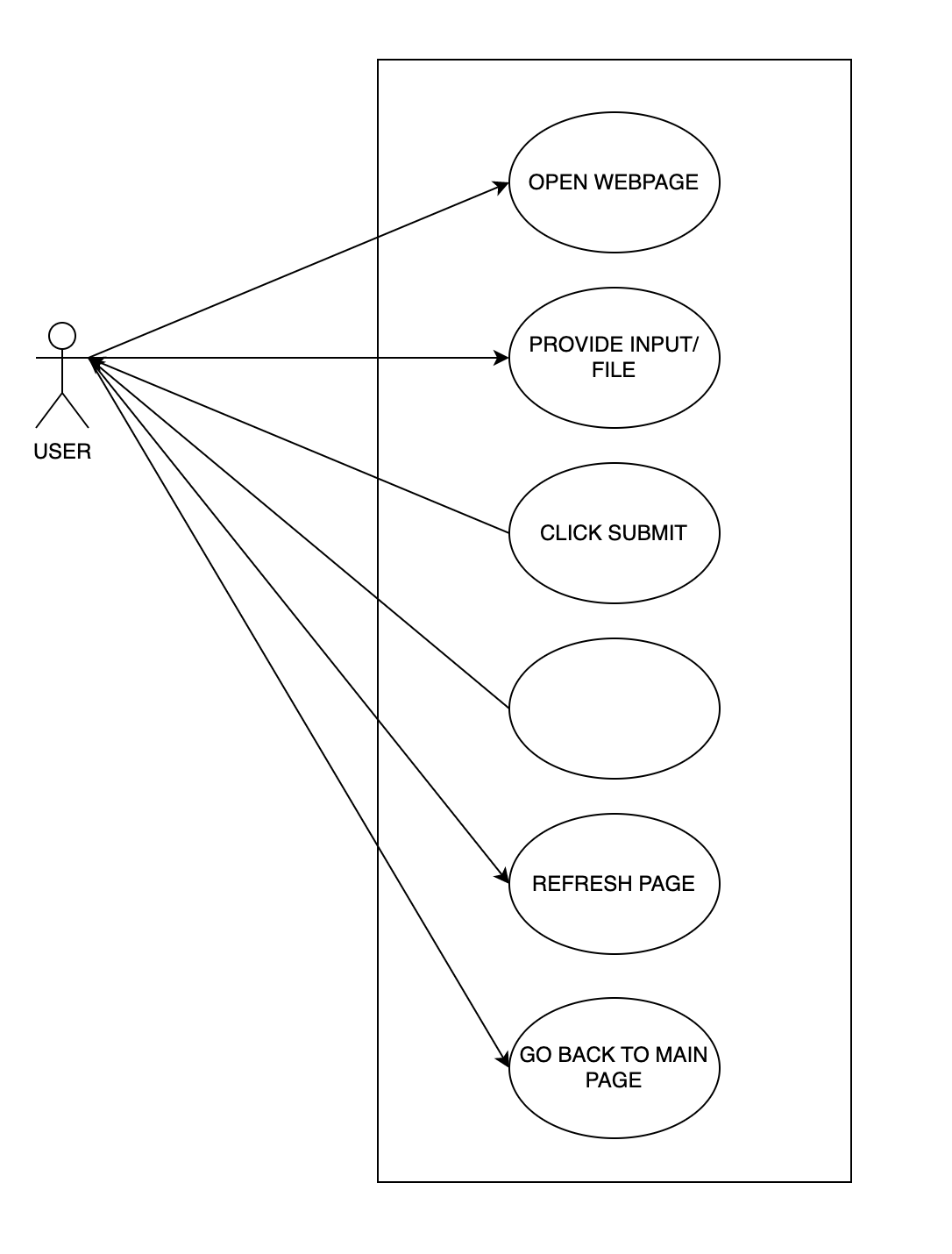
Necessary Python Modules

**Activity diagram:**



OF PLANT TYPE & DETAILS

**Use-case diagram:**

****

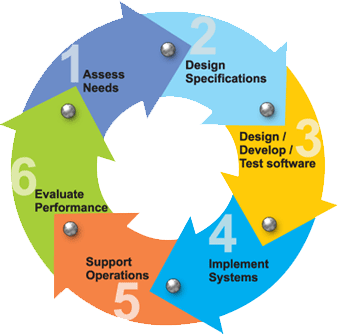
GET PLANT TYPE  
AND DETAILS

SYSTEM ANALYSIS

**2.1 INTRODUCTION**

**Software Development Life Cycle:-**

There is various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred as "Software Development Process Models". Each process model follows a particular life cycle in order to ensure success in process of software development.



**Requirements:-**

Business requirements are gathered in this phase.  This phase is the main focus of the project managers and stake holders.  Meetings with managers, stake holders and users are held in order to determine the requirements.  Who is going to use the system?  How will they use the system?  What data should be input into the system?  What data should be output by the system?  These are general questions that get answered during a requirements gathering phase.  This produces a nice big list of functionality that the system should provide, which describes functions the system should perform, business logic that processes data, what data is stored and used by the system, and how the user interface should work.  The overall result is the system as a whole and how it performs, not how it is actually going to do it.

**Design**

The software system design is produced from the results of the requirements phase.  Architects have the ball in their court during this phase and this is the phase in which their focus lies.  This is where the details on how the system will work is produced.  Architecture, including hardware and software, communication, software design (UML is produced here) are all part of the deliverables of a design phase.

**Implementation**

Code is produced from the deliverables of the design phase during implementation, and this is the longest phase of the software development life cycle.  For a developer, this is the main focus of the life cycle because this is where the code is produced.  Implementation my overlap with both the design and testing phases.  Many tools exists (CASE tools) to actually automate the production of code using information gathered and produced during the design phase.

**Testing**

During testing, the implementation is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase.  Unit tests and system/acceptance tests are done during this phase.  Unit tests act on a specific component of the system, while system tests act on the system as a whole.

So in a nutshell, that is a very basic overview of the general software development life cycle model.  Now let’s delve into some of the traditional and widely used variations.

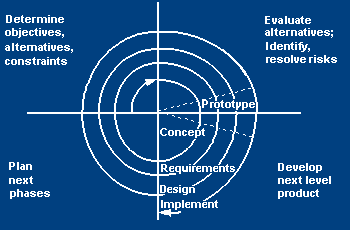
**SDLC METHDOLOGIES**

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

**SPIRAL MODEL** was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

**The following diagram shows how a spiral model acts like:**



The steps for Spiral Model can be generalized as follows:

* The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:

1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
2. Defining the requirements of the second prototype.
3. Planning an designing the second prototype.
4. Constructing and testing the second prototype.

* At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involved development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.
* The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype.
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

**2.2 STUDY OF THE SYSTEM**

In the flexibility of uses the interface has been developed a graphics concepts in mind, associated through a browser interface. The GUI’s at the top level has been categorized as follows

1. Administrative User Interface Design
2. The Operational and Generic User Interface Design

The administrative user interface concentrates on the consistent information that is practically, part of the organizational activities and which needs proper authentication for the data collection. The Interface helps the administration with all the transactional states like data insertion, data deletion, and data updating along with executive data search capabilities.

The operational and generic user interface helps the users upon the system in transactions through the existing data and required services. The operational user interface also helps the ordinary users in managing their own information helps the ordinary users in managing their own information in a customized manner as per the assisted flexibilities.

FEASIBILITY STUDY

INTRODUCTION

Preliminary investigation examines project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

* Technical Feasibility
* Operational Feasibility
* Economical Feasibility

**3.1. TECHNICAL FEASIBILITY**

The technical issue usually raised during the feasibility stage of the investigation includes the following:

* Does the necessary technology exist to do what is suggested?
* Do the proposed equipment have the technical capacity to hold the data required to use the new system?
* Will the proposed system provide adequate response to inquiries, regardless of the number or location of users?
* Can the system be upgraded if developed?
* Are there technical guarantees of accuracy, reliability, ease of access and data security?

Earlier no system existed to cater to the needs of ‘Secure Infrastructure Implementation System’. The current system developed is technically feasible. It is a web-based user interface for audit workflow at NIC-CSD. Thus, it provides an easy access to the users. The database’s purpose is to create, establish and maintain a workflow among various entities in order to facilitate all concerned users in their various capacities or roles. Permission to the users would be granted based on the roles specified. Therefore, it provides the technical guarantee of accuracy, reliability and security. The software and hard requirements for the development of this project are not many and are already available in-house at NIC or are available as free as open source. The work for the project is done with the current equipment and existing software technology. Necessary bandwidth exists for providing a fast feedback to the users irrespective of the number of users using the system.

**3.2. OPERATIONAL FEASIBILITY**

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization’s operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. Some of the important issues raised are to test the operational feasibility of a project includes the following: -

* Is there sufficient support for the management from the users?
* Will the system be used and work properly if it is being developed and implemented?
* Will there be any resistance from the user that will undermine the possible application benefits?

This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So, there is no question of resistance from the users that can undermine the possible application benefits.

The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

**3.3. ECONOMICAL FEASIBILITY**

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economic feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs.

The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at NIC, there is nominal expenditure and economic feasibility for certain.

**FEASIBILITY STUDY:**

Preliminary investigation examines project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

* Technical Feasibility
* Economic Feasibility
* Operation Feasibility

**Technical Feasibility:**

In the feasibility study first step is that the organization or company has to decide that what technologies are suitable to develop by considering existing system.

Here in this application used the technologies like Visual Studio 2012 and SqlServer 2014. These are free software that would be downloaded from web.

Visual Studio 2013 –it is tool or technology.

### Determining Economic Feasibility:

Assessing the economic feasibility of an implementation by performing a cost/benefit analysis, which as its name suggests compares the full/real costs of the application to its full/real financial benefits.  The alternatives should be evaluated on the basis of their contribution to net cash flow, the amount by which the benefits exceed the costs, because the primary objective of all investments is to improve overall organizational performance.

|  |  |  |
| --- | --- | --- |
| **Type** | **Potential Costs** | **Potential Benefits** |
| Quantitative | * Hardware/software upgrades * Fully-burdened cost of labor (salary + benefits) * Support costs for the application * Expected operational costs * Training costs for users to learn the application * Training costs to train developers in new/updated technologies | * Reduced operating costs * Reduced personnel costs from a reduction in staff * Increased revenue from additional sales of your organizations products/services |
| Qualitative | * Increased employee dissatisfaction from fear of change | * Improved decisions as the result of access to accurate and timely information * Raising of existing, or introduction of a new, barrier to entry within your industry to keep competition out of your market * Positive public perception that your organization is an innovator |

 The table includes both qualitative factors, costs or benefits that are subjective in nature, and quantitative factors, costs or benefits for which monetary values can easily be identified.  I will discuss the need to take both kinds of factors into account when performing a cost/benefit analysis.

**Operational Feasibility**

Not only must an application make economic and technical sense, it must also make operational sense.

|  |  |
| --- | --- |
| **Operations Issues** | **Support Issues** |
| * What tools are needed to support operations? * What skills will operators need to be trained in? * What processes need to be created and/or updated? * What documentation does operations need? | * What documentation will users be given? * What training will users be given? * How will change requests be managed? |

Very often you will need to improve the existing operations, maintenance, and support infrastructure to support the operation of the new application that you intend to develop.  To determine what the impact will be you will need to understand both the current operations and support infrastructure of your organization and the operations and support characteristics of your new application. To operate this application END-TO-END VMS. The user no need to require any technical knowledge that we are used to develop this project is Python Programming. That the application providing rich user interface by user can do the operation in flexible manner.

# **Python**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural. It also has a comprehensive standard library. The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.



Python is simple to use, but it is a real programming language, offering much more structure and support for large programs than shell scripts or batch files can offer. On the other hand, Python also offers much more error checking than C, and, being a very-high-level language, it has high-level data types built in, such as flexible arrays and dictionaries. Because of its more general data types Python is applicable to a much larger problem domain than Awk or even Perl, yet many things are at least as easy in Python as in those languages.

Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs — or as examples to start learning to program in Python. Some of these modules provide things like file I/O, system calls, sockets, and even interfaces to graphical user interface toolkits like Tk.

Python is an interpreted language, which can save you considerable time during program development because no compilation and linking is necessary. The interpreter can be used interactively, which makes it easy to experiment with features of the language, to write throw-away programs, or to test functions during bottom-up program development. It is also a handy desk calculator.

Python enables programs to be written compactly and readably. Programs written in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons:

* the high-level data types allow you to express complex operations in a single statement;
* statement grouping is done by indentation instead of beginning and ending brackets;
* no variable or argument declarations are necessary.

Python is extensible: if you know how to program in C it is easy to add a new built-in function or module to the interpreter, either to perform critical operations at maximum speed, or to link Python programs to libraries that may only be available in binary form (such as a vendor-specific graphics library). Once you are really hooked, you can link the Python interpreter into an application written in C and use it as an extension or command language for that application.

Invoking the Interpreter

The Python interpreter is usually installed as /usr/local/bin/python3.7 on those machines where it is available; putting /usr/local/bin in your Unix shell’s search path makes it possible to start it by typing the command:

python3.7

to the shell. [[1]](https://docs.python.org/3.7/tutorial/interpreter.html#id2) Since the choice of the directory where the interpreter lives is an installation option, other places are possible; check with your local Python guru or system administrator. (E.g., /usr/local/python is a popular alternative location.)

On Windows machines, the Python installation is usually placed in C:\Python37, though you can change this when you’re running the installer. To add this directory to your path, you can type the following command into the command prompt in a DOS box:

set path=%path%;C:\python37

Typing an end-of-file character (Control-D on Unix, Control-Z on Windows) at the primary prompt causes the interpreter to exit with a zero exit status. If that doesn’t work, you can exit the interpreter by typing the following command: quit().

The interpreter’s line-editing features include interactive editing, history substitution and code completion on systems that support readline. Perhaps the quickest check to see whether command line editing is supported is typing Control-P to the first Python prompt you get. If it beeps, you have command line editing; see Appendix Interactive Input Editing and History Substitution for an introduction to the keys. If nothing appears to happen, or if ^P is echoed, command line editing isn’t available; you’ll only be able to use backspace to remove characters from the current line.

The interpreter operates somewhat like the Unix shell: when called with standard input connected to a tty device, it reads and executes commands interactively; when called with a file name argument or with a file as standard input, it reads and executes a script from that file.

A second way of starting the interpreter is python -c command [arg] ..., which executes the statement(s) in command, analogous to the shell’s [-c](https://docs.python.org/3.7/using/cmdline.html#cmdoption-c) option. Since Python statements often contain spaces or other characters that are special to the shell, it is usually advised to quote command in its entirety with single quotes.

Some Python modules are also useful as scripts. These can be invoked using python -m module [arg] ..., which executes the source file for module as if you had spelled out its full name on the command line.

When a script file is used, it is sometimes useful to be able to run the script and enter interactive mode afterwards. This can be done by passing [-i](https://docs.python.org/3.7/using/cmdline.html#cmdoption-i) before the script.

**Argument Passing**

When known to the interpreter, the script name and additional arguments thereafter are turned into a list of strings and assigned to the argv variable in the sys module. You can access this list by executing import sys. The length of the list is at least one; when no script and no arguments are given, sys.argv[0] is an empty string. When the script name is given as '-' (meaning standard input), sys.argv[0] is set to '-'. When [-c](https://docs.python.org/3.7/using/cmdline.html#cmdoption-c)command is used, sys.argv[0] is set to '-c'. When [-m](https://docs.python.org/3.7/using/cmdline.html#cmdoption-m) module is used, sys.argv[0] is set to the full name of the located module. Options found after [-c](https://docs.python.org/3.7/using/cmdline.html#cmdoption-c) command or [-m](https://docs.python.org/3.7/using/cmdline.html#cmdoption-m) module are not consumed by the Python interpreter’s option processing but left in sys.argv for the command or module to handle.

**Interactive Mode**

When commands are read from a tty, the interpreter is said to be in interactive mode. In this mode it prompts for the next command with the primary prompt, usually three greater-than signs (>>>); for continuation lines it prompts with the secondary prompt, by default three dots (...). The interpreter prints a welcome message stating its version number and a copyright notice before printing the first prompt:

$ python3.7

Python 3.7 (default, Sep 16 2015, 09:25:04)

[GCC 4.8.2] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>>

Continuation lines are needed when entering a multi-line construct. As an example, take a look at this [if](https://docs.python.org/3.7/reference/compound_stmts.html#if) statement:

>>>

>>> the\_world\_is\_flat = True

>>> if the\_world\_is\_flat:

... print("Be careful not to fall off!")

...

Be careful not to fall off!

In the following examples, input and output are distinguished by the presence or absence of prompts ([>>>](https://docs.python.org/3.7/glossary.html#term) and […](https://docs.python.org/3.7/glossary.html#term-1)): to repeat the example, you must type everything after the prompt, when the prompt appears; lines that do not begin with a prompt are output from the interpreter. Note that a secondary prompt on a line by itself in an example means you must type a blank line; this is used to end a multi-line command.

Many of the examples in this manual, even those entered at the interactive prompt, include comments. Comments in Python start with the hash character, #, and extend to the end of the physical line. A comment may appear at the start of a line or following whitespace or code, but not within a string literal. A hash character within a string literal is just a hash character. Since comments are to clarify code and are not interpreted by Python, they may be omitted when typing in examples.

**Some examples:**

# this is the first comment

spam = 1 # and this is the second comment

# ... and now a third!

text = "# This is not a comment because it's inside quotes."

Python uses [duck typing](https://en.wikipedia.org/wiki/Duck_typing) and has typed objects but untyped variable names. Type constraints are not checked at [compile time](https://en.wikipedia.org/wiki/Compile_time); rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being [dynamically typed](https://en.wikipedia.org/wiki/Type_system#Dynamic_type_checking_and_runtime_type_information), Python is [strongly typed](https://en.wikipedia.org/wiki/Strongly_typed_programming_language), forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using [classes](https://en.wikipedia.org/wiki/Class_(computer_science)), which are most often used for [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming). New [instances](https://en.wikipedia.org/wiki/Object_(computer_science)) of classes are constructed by calling the class (for example, SpamClass() or EggsClass()), and the classes are instances of the [metaclass](https://en.wikipedia.org/wiki/Metaclass" \o "Metaclass) type (itself an instance of itself), allowing [metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming) and [reflection](https://en.wikipedia.org/wiki/Reflection_(computer_science)).

Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long term plan is to support [gradual typing](https://en.wikipedia.org/wiki/Gradual_typing)[[77]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-77) and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named mypy supports compile-time type checking.

Python has the usual C language arithmetic operators (+, -, \*, /, %). It also has \*\* for exponentiation, e.g. 5\*\*3 == 125 and 9\*\*0.5 == 3.0, and a new matrix multiply @ operator is included in version 3.5.[[80]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-80) Additionally, it has a unary operator (~), which essentially inverts all the bits of its one argument. For integers, this means ~x=-x-1.[[81]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-81) Other operators include bitwise shift operators x << y, which shifts x to the left y places, the same as x\*(2\*\*y) , and x >> y, which shifts x to the right y places, the same as x//(2\*\*y) .[[82]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-82)

The behavior of division has changed significantly over time:[[83]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-pep0238-83)[[why?](https://en.wikipedia.org/wiki/Wikipedia:Please_clarify)]

* Python 2.1 and earlier use the C division behavior. The / operator is integer division if both operands are integers, and floating-point division otherwise. Integer division rounds towards 0, e.g. 7/3 == 2 and -7/3 == -2.
* Python 2.2 changes integer division to round towards negative infinity, e.g. 7/3 == 2 and -7/3 == -3. The floor division // operator is introduced. So 7//3 == 2, -7//3 == -3, 7.5//3 == 2.0 and -7.5//3 == -3.0. Adding from \_\_future\_\_ import division causes a module to use Python 3.0 rules for division (see next).
* Python 3.0 changes / to be always floating-point division. In Python terms, the pre-3.0 / is classic division, the version-3.0 / is real division, and // is floor division.

Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it means that the equation (a + b)//b == a//b + 1 is always true. It also means that the equation b\*(a//b) + a%b == a is valid for both positive and negative values of a. However, maintaining the validity of this equation means that while the result of a%b is, as expected, in the [half-open interval](https://en.wikipedia.org/wiki/Half-open_interval) [0, b), where b is a positive integer, it has to lie in the interval (b, 0] when b is negative.

Python provides a round function for [rounding](https://en.wikipedia.org/wiki/Rounding) a float to the nearest integer. For [tie-breaking](https://en.wikipedia.org/wiki/Rounding#Tie-breaking), versions before 3 use round-away-from-zero: round(0.5) is 1.0, round(-0.5) is −1.0. Python 3 uses [round to even](https://en.wikipedia.org/wiki/Round_to_even): round(1.5) is 2, round(2.5) is 2.

Python allows boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression a < b < c tests whether a is less than b and b is less than c. C-derived languages interpret this expression differently: in C, the expression would first evaluate a < b, resulting in 0 or 1, and that result would then be compared with c.

Python has extensive built-in support for [arbitrary precision arithmetic](https://en.wikipedia.org/wiki/Arbitrary_precision_arithmetic). Integers are transparently switched from the machine-supported maximum fixed-precision (usually 32 or 64 bits), belonging to the python type int, to arbitrary precision, belonging to the Python type long, where needed. The latter have an "L" suffix in their textual representation. (In Python 3, the distinction between the int and long types was eliminated; this behavior is now entirely contained by the int class.) The Decimal type/class in module decimal (since version 2.4) provides decimal floating point numbers to arbitrary precision and several rounding modes. The Fraction type in module fractions (since version 2.6) provides arbitrary precision for rational numbers.

Due to Python's extensive mathematics library, and the third-party library [NumPy](https://en.wikipedia.org/wiki/NumPy) that further extends the native capabilities, it is frequently used as a scientific scripting language to aid in problems such as numerical data processing and manipulation.

**How to Install Python on Windows**



Python doesn’t come prepackaged with Windows, but that doesn’t mean Windows users won’t find the flexible programming language useful. It’s not quite a simple as installing the newest version however, so let’s make sure you get the right tools for the task at hand.

First released in 1991, Python is a popular high-level programming language used for general purpose programming. Thanks to a design philosophy that emphasizes readability it has long been a favorite of hobby coders and serious programmers alike. Not only is it an easy language (comparatively speaking, that is) to pick up but you’ll find thousands of projects online that require you have Python installed to use the program.

**Which Version Do You Need?**

Unfortunately, there was a significant update to Python several years ago that created a big split between Python versions. This can make things a bit confusing to newcomers, but don’t worry. We’ll walk you through installing both major versions

When you visit the [Python for Windows download page](https://www.python.org/downloads/windows/), you’ll immediately see the division. Right at the top, square and center, the repository asks if you want the latest release of Python 2 or Python 3 (2.7.13 and 3.6.1, respectively, as of this tutorial).



Newer is better, right? Maybe so, maybe not. The version you want depends on your end goal. That project is coded in Python and requires Python 2.7—you can’t run the MCDungeon project with Python 3.6. In fact, if you’re exploring hobby projects like MCDungeon, you’ll find that nearly all of them use 2.7. If your goal is to get some project that ends in a “.py” extension up and running, then there’s a very, *very* good chance you’ll need 2.7 for it.

On the other hand, if you’re looking to actually learn Python, we recommend installing both versions side by side (which you can do with zero risk and only a tiny bit of setup hassle). This lets you work with the newest version of the language, but also run older Python scripts (and test backwards compatibility for newer projects). Comparing the two versions is an article unto itself, though, so we’ll defer to the Python project wiki where you can read their well written overview of the differences.

You can download just Python 2 or Python 3 if you’re sure you only need a particular version. We’re going the distance today and will be installing both of them, so we recommend you download both versions and do the same. Under the main entry for both versions you’ll see an “x86-64” installer, as seen below.

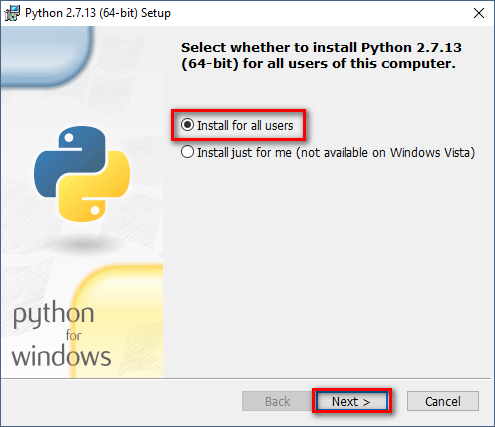


**RELATED:** [***What's the Difference Between 32-bit and 64-bit Windows?***](https://www.howtogeek.com/56701/htg-explains-whats-the-difference-between-32-bit-and-64-bit-windows-7/)

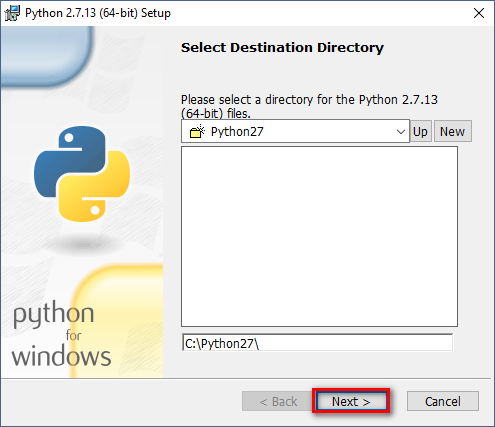
This installer will install the appropriate 32-bit or 64-bit version on your computer automatically (here’s [some further reading](https://www.howtogeek.com/56701/htg-explains-whats-the-difference-between-32-bit-and-64-bit-windows-7/) if you want to know more about the differences between the two).

**How to Install Python 2**

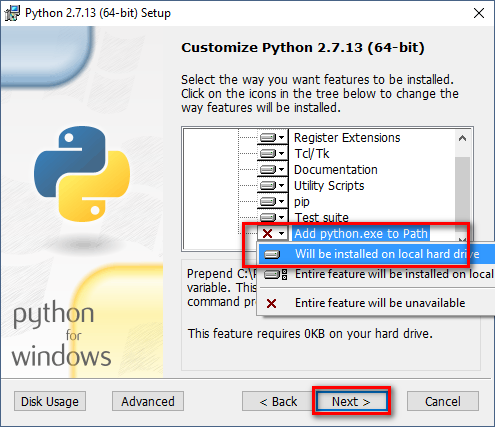
Installing Python 2 is a snap, and unlike in years past, the installer will even set the path variable for you (something we’ll be getting into a bit later). [Download and run the installer](https://www.python.org/downloads/windows/), select “Install for all users,” and then click “Next.”



On the directory selection screen, leave the directory as “Python27” and click “Next.”



On the customization screen, scroll down, click “Add python.exe to Path,” and then select “Will be installed on local hard drive.” When you’re done, click “Next.”



You don’t have to make any more decisions after this point. Just click through the wizard to complete the installation. When the installation is finished, you can confirm the installation by opening up Command Prompt and typing the following command:

python -V

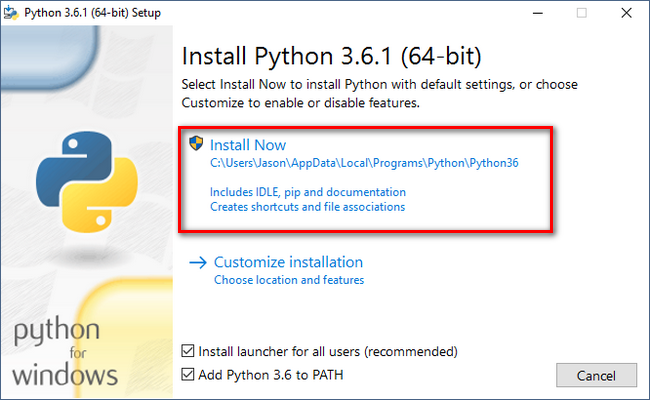
/var/folders/hc/ftm75lgd19l2jw74w31dd8s40000gn/T/com.microsoft.Word/Content.MSO/F6ECA7E4.tmp

Success! If all you need is Python 2.7 for some project or another, you can stop right here. It’s installed, the path variable is set, and you’re off to the races.

**How to Install Python 3**

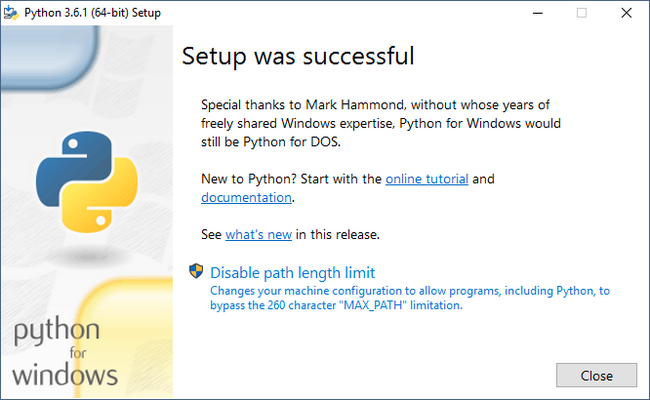
If you want to learn the newest version of Python, you’ll need to install Python 3. You can install it alongside Python 2.7 with no problems, so go ahead and [download and run the installer](https://www.python.org/downloads/windows/) now.

On the first screen, enable the “Add Python 3.6 to PATH” option and then click “Install Now.”



Next, you have a decision to make. Clicking the “Disable path length limit” option removes the limitation on the MAX\_PATH variable. This change won’t break anything, but will allow Python to use long path names. Since many Python programmers are working in Linux and other \*nix systems where path name length isn’t an issue, turning this on in advance can help smooth over any path-related issues you might have while working in Windows.

We recommend go ahead and selecting this option. If you know you don’t want to disable the path length limit, you can just click “Close” to finish the installation. And, if you want to read more about the issue before committing to the change, [read up here](https://www.howtogeek.com/266621/how-to-make-windows-10-accept-file-paths-over-260-characters/).



If you’re only installing Python 3, you can use the same command line trick of typing python -v that we used above to check that it is installed correctly and the path variable is set. If you’re installing both versions, however, you need to make the quick tweak found in the following section.

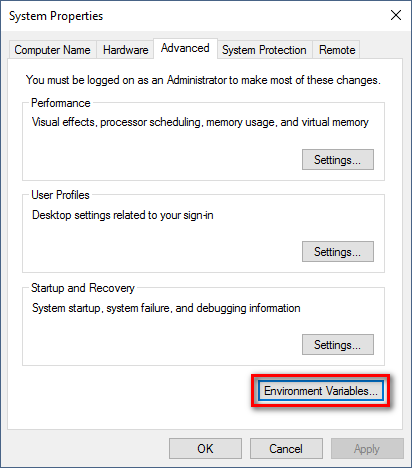
**Adjust System Variables So You Can Access Both Python Versions From the Command Line**

This section of the tutorial is completely optional, but will allow you to quickly access both versions of Python from the command line. After installing both versions of Python, you may have noticed a little quirk. Even though we enabled the system path for both Python installations, typing “python” at the command prompt only points you to Python 2.7.

The reason for this is simple: the variable (whether automatically adjusted by an installer or manually tweaked) simply points at a directory, and every executable in that directory becomes a command line command. If there are two directories listed and both have a “python.exe” file in them, whichever directory is higher in the list of variables gets used. And, if there is a variable set for the system and the user, the system path takes precedence over the user path.

The latter is exactly what’s happening in this case: the Python 2 installer edited the system wide variable and the Python 3 installer added a user level variable—and we can confirm this by looking at the Windows’ environment variables.

Hit Start, type “advanced system settings,” and then select the “View advanced system settings” option. In the “System Properties” window that opens, on the “Advanced” tab, click the “Environment Variables” button.

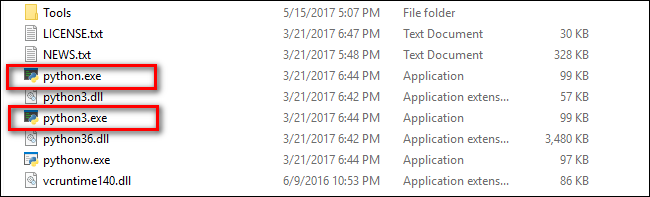


Here, you can see Python 3 listed in the “User variables” section and Python 2 listed in the “System variables” section.



There are a few ways you can remedy this situation. The simplest (albeit the one with the least functionality) is to just remove the entry for the version of Python you plan on using the least. While that’s simple, it’s also not very much fun. Instead we can make another change that will give us access to “python” for Python 2 and “python3” for Python 3.

To do this, fire up File Manager and head to the folder where you installed Python 3 (C:\Users\[username]\AppData\Local\Programs\Python\Python36 by default). Make a copy of the “python.exe” file, and rename that copy (*not* the original) to “python3.exe”.



Open a new command prompt (the environmental variables refresh with each new command prompt you open), and type “python3 –version”.

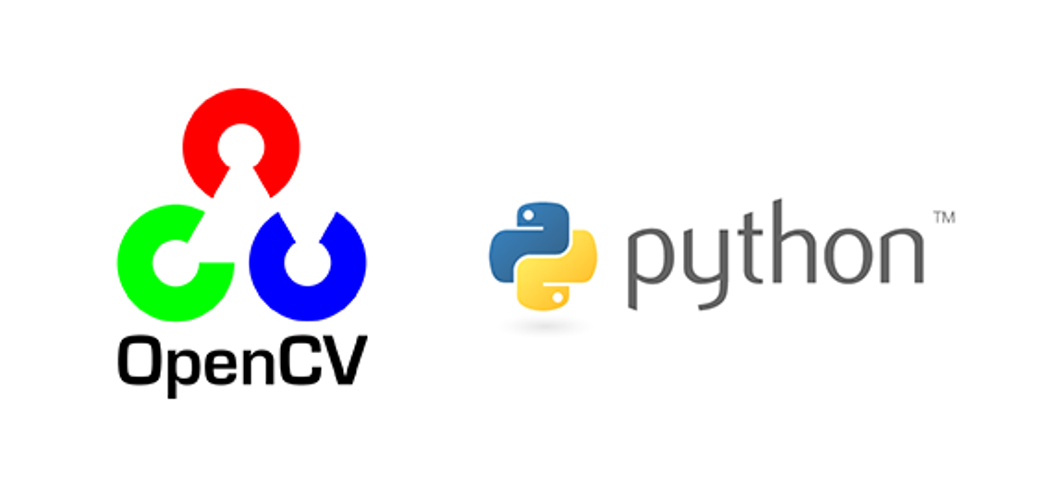
/var/folders/hc/ftm75lgd19l2jw74w31dd8s40000gn/T/com.microsoft.Word/Content.MSO/C9F618E7.tmp

Boom! You can now use the “python” command at the Command Prompt when you want to use Python 2.7 and the “python3” command when you want to use Python 3.

If, for whatever reason, you don’t find this a satisfactory solution, you can always reorder the environmental variables. Be sure to [brush up with our tutorial](https://www.howtogeek.com/118594/how-to-edit-your-system-path-for-easy-command-line-access/) first if you’re not comfortable editing those variables.

Please note, however, that regardless of which method you use it is important to leave the original python.exe intact as the applications in the /scripts/ subdirectory for both versions of Python rely on that filename and will fail if it is missing.

# **Opencv**



OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV’s deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDAand OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

* Core functionality (core) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
* Image Processing (imgproc) - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
* Video Analysis (video) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
* Camera Calibration and 3D Reconstruction (calib3d) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
* 2D Features Framework (features2d) - salient feature detectors, descriptors, and descriptor matchers.
* Object Detection (objdetect) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
* High-level GUI (highgui) - an easy-to-use interface to simple UI capabilities.
* Video I/O (videoio) - an easy-to-use interface to video capturing and video codecs.
* ... some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

OpenCV was started at Intel in 1999 by **Gary Bradsky** and the first release came out in 2000. **Vadim Pisarevsky** joined Gary Bradsky to manage Intel’s Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle who won 2005 DARPA Grand Challenge. Later its active development continued under the support of Willow Garage, with Gary Bradsky and Vadim Pisarevsky leading the project. Right now, OpenCV supports a lot of algorithms related to Computer Vision and Machine Learning and it is expanding day-by-day.

Currently OpenCV supports a wide variety of programming languages like C++, Python, Java etc and is available on different platforms including Windows, Linux, OS X, Android, iOS etc. Also, interfaces based on CUDA and OpenCL are also under active development for high-speed GPU operations.

OpenCV-Python is the Python API of OpenCV. It combines the best qualities of OpenCV C++ API and Python language.

## **OpenCV-Python**

Python is a general purpose programming language started by **Guido van Rossum**, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability.

Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++. This feature helps us to write computationally intensive codes in C/C++ and create a Python wrapper for it so that we can use these wrappers as Python modules. This gives us two advantages: first, our code is as fast as original C/C++ code (since it is the actual C++ code working in background) and second, it is very easy to code in Python. This is how OpenCV-Python works, it is a Python wrapper around original C++ implementation.

And the support of Numpy makes the task more easier. **Numpy** is a highly optimized library for numerical operations. It gives a MATLAB-style syntax. All the OpenCV array structures are converted to-and-from Numpy arrays. So whatever operations you can do in Numpy, you can combine it with OpenCV, which increases number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this.

So OpenCV-Python is an appropriate tool for fast prototyping of computer vision problems

**Automatic Memory Management**

OpenCV handles all the memory automatically.

First of all, std::vector, cv::Mat, and other data structures used by the functions and methods have destructors that deallocate the underlying memory buffers when needed. This means that the destructors do not always deallocate the buffers as in case of Mat. They take into account possible data sharing. A destructor decrements the reference counter associated with the matrix data buffer. The buffer is deallocated if and only if the reference counter reaches zero, that is, when no other structures refer to the same buffer. Similarly, when a Mat instance is copied, no actual data is really copied. Instead, the reference counter is incremented to memorize that there is another owner of the same data. There is also the Mat::clone method that creates a full copy of the matrix data.

**Automatic Allocation of the Output Data**

OpenCV deallocates the memory automatically, as well as automatically allocates the memory for output function parameters most of the time. So, if a function has one or more input arrays (cv::Mat instances) and some output arrays, the output arrays are automatically allocated or reallocated. The size and type of the output arrays are determined from the size and type of input arrays. If needed, the functions take extra parameters that help to figure out the output array properties.

The array frame is automatically allocated by the >> operator since the video frame resolution and the bit-depth is known to the video capturing module. The array edges is automatically allocated by the cvtColor function. It has the same size and the bit-depth as the input array. The number of channels is 1 because the color conversion code cv::COLOR\_BGR2GRAY is passed, which means a color to grayscale conversion. Note that frame and edges are allocated only once during the first execution of the loop body since all the next video frames have the same resolution. If you somehow change the video resolution, the arrays are automatically reallocated.

The key component of this technology is the Mat::create method. It takes the desired array size and type. If the array already has the specified size and type, the method does nothing. Otherwise, it releases the previously allocated data, if any (this part involves decrementing the reference counter and comparing it with zero), and then allocates a new buffer of the required size. Most functions call the Mat::create method for each output array, and so the automatic output data allocation is implemented.

Some notable exceptions from this scheme are cv::mixChannels, cv::RNG::fill, and a few other functions and methods. They are not able to allocate the output array, so you have to do this in advance.

**Saturation Arithmetics**

As a computer vision library, OpenCV deals a lot with image pixels that are often encoded in a compact, 8- or 16-bit per channel, form and thus have a limited value range. Furthermore, certain operations on images, like color space conversions, brightness/contrast adjustments, sharpening, complex interpolation (bi-cubic, Lanczos) can produce values out of the available range. If you just store the lowest 8 (16) bits of the result, this results in visual artifacts and may affect a further image analysis. To solve this problem, the so-called saturation arithmetics is used. For example, to store r, the result of an operation, to an 8-bit image, you find the nearest value within the 0..255 range:

**I(x,y)=min(max(round(r),0),255)**

Similar rules are applied to 8-bit signed, 16-bit signed and unsigned types. This semantics is used everywhere in the library. In C++ code, it is done using the cv::saturate\_cast<> functions that resemble standard C++ cast operations.

**Fixed Pixel Types. Limited Use of Templates**

Templates is a great feature of C++ that enables implementation of very powerful, efficient and yet safe data structures and algorithms. However, the extensive use of templates may dramatically increase compilation time and code size. Besides, it is difficult to separate an interface and implementation when templates are used exclusively. This could be fine for basic algorithms but not good for computer vision libraries where a single algorithm may span thousands lines of code. Because of this and also to simplify development of bindings for other languages, like Python, Java, Matlab that do not have templates at all or have limited template capabilities, the current OpenCV implementation is based on polymorphism and runtime dispatching over templates. In those places where runtime dispatching would be too slow (like pixel access operators), impossible (generic cv::Ptr<> implementation), or just very inconvenient (cv::saturate\_cast<>()) the current implementation introduces small template classes, methods, and functions. Anywhere else in the current OpenCV version the use of templates is limited.

Consequently, there is a limited fixed set of primitive data types the library can operate on. That is, array elements should have one of the following types:

* 8-bit unsigned integer (uchar)
* 8-bit signed integer (schar)
* 16-bit unsigned integer (ushort)
* 16-bit signed integer (short)
* 32-bit signed integer (int)
* 32-bit floating-point number (float)
* 64-bit floating-point number (double)
* a tuple of several elements where all elements have the same type (one of the above). An array whose elements are such tuples, are called multi-channel arrays, as opposite to the single-channel arrays, whose elements are scalar values. The maximum possible number of channels is defined by the CV\_CN\_MAX constant, which is currently set to 512.

For these basic types, the following enumeration is applied:

**enum { CV\_8U=0, CV\_8S=1, CV\_16U=2, CV\_16S=3, CV\_32S=4, CV\_32F=5, CV\_64F=6 };**

Multi-channel (n-channel) types can be specified using the following options:

* CV\_8UC1 ... CV\_64FC4 constants (for a number of channels from 1 to 4)
* CV\_8UC(n) ... CV\_64FC(n) or CV\_MAKETYPE(CV\_8U, n) ... CV\_MAKETYPE(CV\_64F, n) macros when the number of channels is more than 4 or unknown at the compilation time.

Examples:

Mat mtx(3, 3, CV\_32F); // make a 3x3 floating-point matrix

Mat cmtx(10, 1, CV\_64FC2); // make a 10x1 2-channel floating-point

// matrix (10-element complex vector)

Mat img(Size(1920, 1080), CV\_8UC3); // make a 3-channel (color) image

// of 1920 columns and 1080 rows.

Mat grayscale(image.size(), CV\_MAKETYPE(image.depth(), 1)); // make a 1-channel image of

// the same size and same

// channel type as img

Arrays with more complex elements cannot be constructed or processed using OpenCV. Furthermore, each function or method can handle only a subset of all possible array types. Usually, the more complex the algorithm is, the smaller the supported subset of formats is. See below typical examples of such limitations:

* The face detection algorithm only works with 8-bit grayscale or color images.
* Linear algebra functions and most of the machine learning algorithms work with floating-point arrays only.
* Basic functions, such as cv::add, support all types.
* Color space conversion functions support 8-bit unsigned, 16-bit unsigned, and 32-bit floating-point types.

The subset of supported types for each function has been defined from practical needs and could be extended in future based on user requests.

### **InputArray and OutputArray**

Many OpenCV functions process dense 2-dimensional or multi-dimensional numerical arrays. Usually, such functions take cppMat as parameters, but in some cases it's more convenient to use std::vector<> (for a point set, for example) or cv::Matx<> (for 3x3 homography matrix and such). To avoid many duplicates in the API, special "proxy" classes have been introduced. The base "proxy" class is cv::InputArray. It is used for passing read-only arrays on a function input. The derived from InputArray class cv::OutputArray is used to specify an output array for a function. Normally, you should not care of those intermediate types (and you should not declare variables of those types explicitly) - it will all just work automatically. You can assume that instead of InputArray/OutputArray you can always use Mat, std::vector<>, cv::Matx<>, cv::Vec<> or cv::Scalar. When a function has an optional input or output array, and you do not have or do not want one, pass cv::noArray().

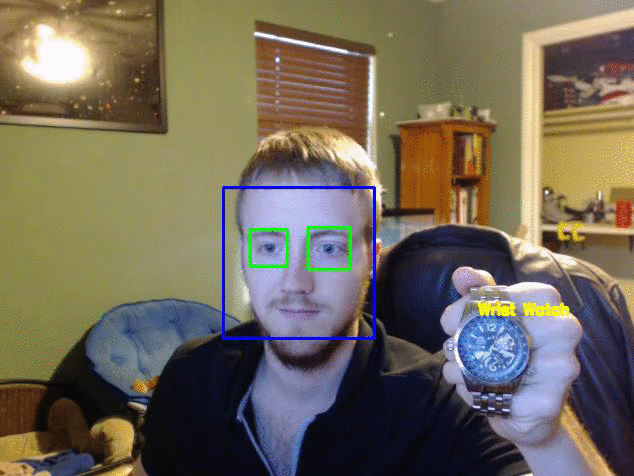
### **Error Handling**

OpenCV uses exceptions to signal critical errors. When the input data has a correct format and belongs to the specified value range, but the algorithm cannot succeed for some reason (for example, the optimization algorithm did not converge), it returns a special error code (typically, just a boolean variable).

The exceptions can be instances of the cv::Exception class or its derivatives. In its turn, cv::Exception is a derivative of std::exception. So it can be gracefully handled in the code using other standard C++ library components.

Multi-threading and Re-enterability

The current OpenCV implementation is fully re-enterable. That is, the same function or the same methods of different class instances can be called from different threads. Also, the same Mat can be used in different threads because the reference-counting operations use the architecture-specific atomic instructions.



Getting started with OpenCV's Python bindings is actually much easier than many people make it out to be initially. You will need two main libraries, with an optional third: python-OpenCV, Numpy, and Matplotlib.

### **Windows Users:**

[**python-OpenCV**](https://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv) - There are alternative methods, but this is the easiest. Download the appropriate wheel (.whl) file, and install using pip. See video for help.

pip install numpy

pip install matplotlib

Not familiar with using pip? See the [**Pip installation tutorial**](https://pythonprogramming.net/using-pip-install-for-python-modules/) for help.

### **Linux / Mac Users:**

pip3 install numpy or apt-get install python3-numpy. You may need to apt-get install python3-pip.

pip3 install matplotlib or apt-get install python3-matplotlib.

apt-get install python3-OpenCV

[**Matplotlib**](https://pythonprogramming.net/matplotlib-intro-tutorial/) is an optional choice for displaying frames from video or images. We will show a couple of examples using it here. **[Numpy](https://www.numpy.org/" \t "blank)** is used for all things "numbers and Python." We are mainly making use of Numpy's array functionality. Finally, we are using the python-specific bindings for OpenCV called python-OpenCV.

There are some operations for OpenCV that you will not be able to do without a full installation of OpenCV (about 3GB in size), but you can actually do quite a bit with the fairly minimal installation of python-OpenCV. We will wind up using the full installation of OpenCV later in this series, so you can feel free to get it if you like, but these 3 modules will keep us busy for a while!

Make sure your installations were successful by running Python, and doing:

import cv2

import matplotlib

import numpy

If you get no errors, then you are ready to go. Ready? Let's dive off the deep-end!

First, we should understand a few basic assumptions and paradigms when it comes to image and video analysis. With the way just about every video camera records today, recordings are actually frames, displayed one after another, 30-60+ times a second. At the core, however, they are static frames, just like images. Thus, image recognition and video analysis use identical methods for the most part. Some things, like directional tracking, is going to require a succession of images (frames), but something like facial detection, or object recognition can be done with almost the exact same code on images and video.

Next, a lot of image and video analysis boils down to simplifying the source as much as possible. This almost always begins with a conversion to grayscale, but it can also be a color filter, gradient, or a combination of these. From here, we can do all sorts of analysis and transformations to the source. Generally, what winds up happening is there is a transformation done, then analysis, then any overlays that we wish to apply are applied back to the original source, which is why you can often see the "finished product" of maybe object or facial recognition being shown on a full-color image or video. Rarely is the data actually processed in raw form like this, however. Some examples of what we can do at a basic level. All of these are done with a basic web cam, nothing special:

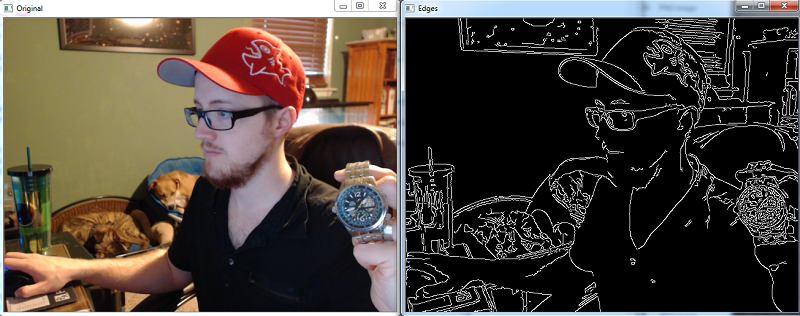
#### **Background Subtracting:**



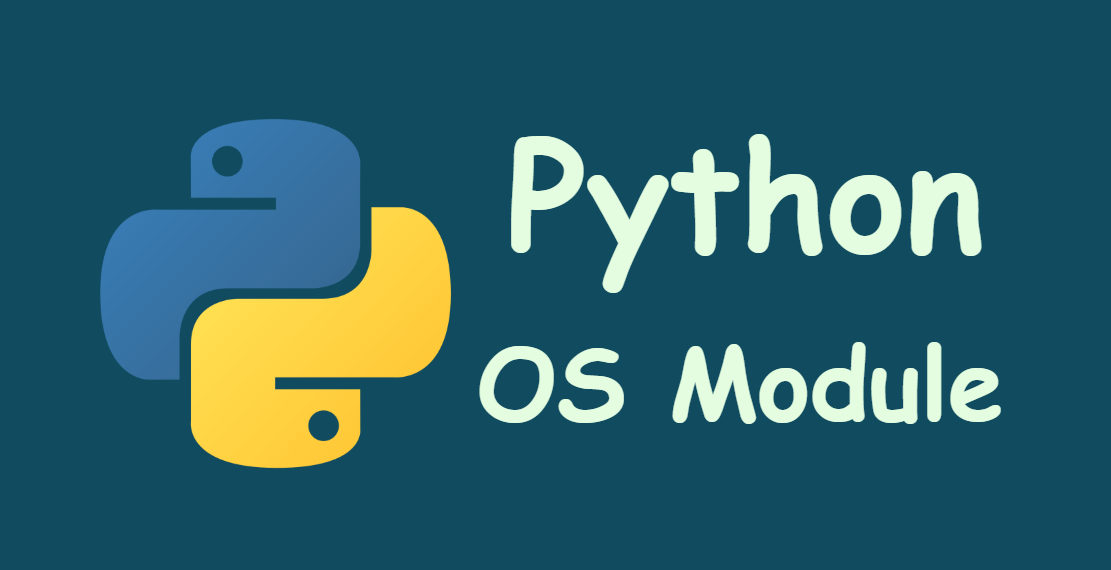
#### **Color filtering:**



#### **Edge detection:**



**OS**



This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For creating temporary files and directories see the tempfile module, and for high-level file and directory handling see the shutil module.

Notes on the availability of these functions:

The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface; for example, the function os.stat(path) returns stat information about path in the same format (which happens to have originated with the POSIX interface).

Extensions peculiar to a particular operating system are also available through the os module, but using them is of course a threat to portability.

All functions accepting path or file names accept both bytes and string objects, and result in an object of the same type, if a path or file name is returned.

Note All functions in this module raise OSError in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system.

exception os.error

An alias for the built-in OSError exception.

**os.name**

The name of the operating system dependent module imported. The following names have currently been registered: 'posix', 'nt', 'java'.

See also sys.platform has a finer granularity. os.uname() gives system-dependent version information.

The platform module provides detailed checks for the system’s identity.

File Names, Command Line Arguments, and Environment Variables

In Python, file names, command line arguments, and environment variables are represented using the string type. On some systems, decoding these strings to and from bytes is necessary before passing them to the operating system. Python uses the file system encoding to perform this conversion (see sys.getfilesystemencoding()).

Changed in version 3.1: On some systems, conversion using the file system encoding may fail. In this case, Python uses the surrogateescape encoding error handler, which means that undecodable bytes are replaced by a Unicode character U+DCxx on decoding, and these are again translated to the original byte on encoding.

The file system encoding must guarantee to successfully decode all bytes below 128. If the file system encoding fails to provide this guarantee, API functions may raise UnicodeErrors.

**Process Parameters**

These functions and data items provide information and operate on the current process and user.

**os.ctermid()**

Return the filename corresponding to the controlling terminal of the process.

Availability: Unix.

**os.environ**

A mapping object representing the string environment. For example, environ['HOME'] is the pathname of your home directory (on some platforms), and is equivalent to getenv("HOME") in C.

This mapping is captured the first time the os module is imported, typically during Python startup as part of processing site.py. Changes to the environment made after this time are not reflected in os.environ, except for changes made by modifying os.environ directly.

If the platform supports the putenv() function, this mapping may be used to modify the environment as well as query the environment. putenv() will be called automatically when the mapping is modified.

On Unix, keys and values use sys.getfilesystemencoding() and 'surrogateescape' error handler. Use environb if you would like to use a different encoding.

Note Calling putenv() directly does not change os.environ, so it’s better to modify os.environ.

Note On some platforms, including FreeBSD and Mac OS X, setting environ may cause memory leaks. Refer to the system documentation for putenv().

If putenv() is not provided, a modified copy of this mapping may be passed to the appropriate process-creation functions to cause child processes to use a modified environment.

If the platform supports the unsetenv() function, you can delete items in this mapping to unset environment variables. unsetenv() will be called automatically when an item is deleted from os.environ, and when one of the pop() or clear() methods is called.

**os.environb**

Bytes version of environ: a mapping object representing the environment as byte strings. environ and environb are synchronized (modify environb updates environ, and vice versa).

environb is only available if supports\_bytes\_environ is True.

New in version 3.2.

**os.chdir(path)**

**os.fchdir(fd)**

**os.getcwd()**

These functions are described in Files and Directories.

**os.fsencode(filename)**

Encode path-like filename to the filesystem encoding with 'surrogateescape' error handler, or 'strict' on Windows; return bytes unchanged.

**fsdecode() is the reverse function.**

**os.fsdecode(filename)**

Decode the path-like filename from the filesystem encoding with 'surrogateescape' error handler, or 'strict' on Windows; return str unchanged.

fsencode() is the reverse function.

Changed in version 3.6: Support added to accept objects implementing the os.PathLike interface.

**os.fspath(path)**

Return the file system representation of the path.

If str or bytes is passed in, it is returned unchanged. Otherwise \_\_fspath\_\_() is called and its value is returned as long as it is a str or bytes object. In all other cases, TypeError is raised.

**class os.PathLike**

An abstract base class for objects representing a file system path, e.g. pathlib.PurePath.

**abstractmethod \_\_fspath\_\_()**

Return the file system path representation of the object.

The method should only return a str or bytes object, with the preference being for str.

**os.getenv(key, default=None)**

Return the value of the environment variable key if it exists, or default if it doesn’t. key, default and the result are str.

On Unix, keys and values are decoded with sys.getfilesystemencoding() and 'surrogateescape' error handler. Use os.getenvb() if you would like to use a different encoding.

Availability: most flavors of Unix, Windows.

**os.getenvb(key, default=None)**

Return the value of the environment variable key if it exists, or default if it doesn’t. key, default and the result are bytes.

**getenvb()** is only available if supports\_bytes\_environ is True.

Availability: most flavors of Unix.

**os.get\_exec\_path(env=None)**

Returns the list of directories that will be searched for a named executable, similar to a shell, when launching a process. env, when specified, should be an environment variable dictionary to lookup the PATH in. By default, when env is None, environ is used.

**os.getegid()**

Return the effective group id of the current process. This corresponds to the “set id” bit on the file being executed in the current process.

Availability: Unix.

**os.geteuid()**

Return the current process’s effective user id.

Availability: Unix.

**os.getgid()**

Return the real group id of the current process.

Availability: Unix.

**os.getgrouplist(user, group)**

Return list of group ids that user belongs to. If group is not in the list, it is included; typically, group is specified as the group ID field from the password record for user.

Availability: Unix.

**os.getgroups()**

Return list of supplemental group ids associated with the current process.

Availability: Unix.

Note On Mac OS X, getgroups() behavior differs somewhat from other Unix platforms. If the Python interpreter was built with a deployment target of 10.5 or earlier, getgroups() returns the list of effective group ids associated with the current user process; this list is limited to a system-defined number of entries, typically 16, and may be modified by calls to setgroups() if suitably privileged. If built with a deployment target greater than 10.5, getgroups() returns the current group access list for the user associated with the effective user id of the process; the group access list may change over the lifetime of the process, it is not affected by calls to setgroups(), and its length is not limited to 16. The deployment target value, MACOSX\_DEPLOYMENT\_TARGET, can be obtained with sysconfig.get\_config\_var().

os.getlogin()

Return the name of the user logged in on the controlling terminal of the process. For most purposes, it is more useful to use getpass.getuser() since the latter checks the environment variables LOGNAME or USERNAME to find out who the user is, and falls back to pwd.getpwuid(os.getuid())[0] to get the login name of the current real user id.

Availability: Unix, Windows.

**os.getpgid(pid)**

Return the process group id of the process with process id pid. If pid is 0, the process group id of the current process is returned.

Availability: Unix.

**os.getpgrp()**

Return the id of the current process group.

Availability: Unix.

**os.getpid()**

Return the current process id.

**os.getppid()**

Return the parent’s process id. When the parent process has exited, on Unix the id returned is the one of the init process (1), on Windows it is still the same id, which may be already reused by another process.

Availability: Unix, Windows.

Changed in version 3.2: Added support for Windows.

**os.getpriority(which, who)**

Get program scheduling priority. The value which is one of PRIO\_PROCESS, PRIO\_PGRP, or PRIO\_USER, and who is interpreted relative to which (a process identifier for PRIO\_PROCESS, process group identifier for PRIO\_PGRP, and a user ID for PRIO\_USER). A zero value for who denotes (respectively) the calling process, the process group of the calling process, or the real user ID of the calling process.

Availability: Unix.

New in version 3.3.

**os.PRIO\_PROCESS**

**os.PRIO\_PGRP**

**os.PRIO\_USER**

Parameters for the getpriority() and setpriority() functions.

Availability: Unix.

New in version 3.3.

**os.getresuid()**

Return a tuple (ruid, euid, suid) denoting the current process’s real, effective, and saved user ids.

Availability: Unix.

**os.getresgid()**

Return a tuple (rgid, egid, sgid) denoting the current process’s real, effective, and saved group ids.

Availability: Unix.

**os.getuid()**

Return the current process’s real user id.

Availability: Unix.

**os.initgroups(username, gid)**

Call the system initgroups() to initialize the group access list with all of the groups of which the specified username is a member, plus the specified group id.

Availability: Unix.

**os.putenv(key, value)**

Set the environment variable named key to the string value. Such changes to the environment affect subprocesses started with os.system(), popen() or fork() and execv().

Availability: most flavors of Unix, Windows.

Note On some platforms, including FreeBSD and Mac OS X, setting environ may cause memory leaks. Refer to the system documentation for putenv.

When putenv() is supported, assignments to items in os.environ are automatically translated into corresponding calls to putenv(); however, calls to putenv() don’t update os.environ, so it is actually preferable to assign to items of os.environ.

**os.setegid(egid)**

Set the current process’s effective group id.

Availability: Unix.

**os.seteuid(euid)**

Set the current process’s effective user id.

Availability: Unix.

**os.setgid(gid)**

Set the current process’ group id.

Availability: Unix.

**os.setgroups(groups)**

Set the list of supplemental group ids associated with the current process to groups. groups must be a sequence, and each element must be an integer identifying a group. This operation is typically available only to the superuser.

Availability: Unix.

Note On Mac OS X, the length of groups may not exceed the system-defined maximum number of effective group ids, typically 16. See the documentation for getgroups() for cases where it may not return the same group list set by calling setgroups().

**os.setpgrp()**

Call the system call setpgrp() or setpgrp(0, 0) depending on which version is implemented (if any). See the Unix manual for the semantics.

Availability: Unix.

**os.setpgid(pid, pgrp)**

Call the system call setpgid() to set the process group id of the process with id pid to the process group with id pgrp. See the Unix manual for the semantics.

Availability: Unix.

**os.setpriority(which, who, priority)**

Set program scheduling priority. The value which is one of PRIO\_PROCESS, PRIO\_PGRP, or PRIO\_USER, and who is interpreted relative to which (a process identifier for PRIO\_PROCESS, process group identifier for PRIO\_PGRP, and a user ID for PRIO\_USER). A zero value for who denotes (respectively) the calling process, the process group of the calling process, or the real user ID of the calling process. priority is a value in the range -20 to 19. The default priority is 0; lower priorities cause more favorable scheduling.

Availability: Unix.

**os.setregid(rgid, egid)**

Set the current process’s real and effective group ids.

Availability: Unix.

**os.setresgid(rgid, egid, sgid)**

Set the current process’s real, effective, and saved group ids.

Availability: Unix.

**os.setresuid(ruid, euid, suid)**

Set the current process’s real, effective, and saved user ids.

Availability: Unix.

New in version 3.2.

**os.setreuid(ruid, euid)**

Set the current process’s real and effective user ids.

Availability: Unix.

**os.getsid(pid)**

Call the system call getsid(). See the Unix manual for the semantics.

Availability: Unix.

**os.setsid()**

Call the system call setsid(). See the Unix manual for the semantics.

Availability: Unix.

**os.setuid(uid)**

Set the current process’s user id.

Availability: Unix.

**os.strerror(code)**

Return the error message corresponding to the error code in code. On platforms where strerror() returns NULL when given an unknown error number, ValueError is raised.

os.supports\_bytes\_environ

True if the native OS type of the environment is bytes (eg. False on Windows).

New in version 3.2.

**os.umask(mask)**

Set the current numeric umask and return the previous umask.

**os.uname()**

Returns information identifying the current operating system. The return value is an object with five attributes:

sysname - operating system name

nodename - name of machine on network (implementation-defined)

release - operating system release

version - operating system version

machine - hardware identifier

For backwards compatibility, this object is also iterable, behaving like a five-tuple containing sysname, nodename, release, version, and machine in that order.

Some systems truncate nodename to 8 characters or to the leading component; a better way to get the hostname is socket.gethostname() or even socket.gethostbyaddr(socket.gethostname()).

Availability: recent flavors of Unix.

Changed in version 3.3: Return type changed from a tuple to a tuple-like object with named attributes.

**os.unsetenv(key)**

Unset (delete) the environment variable named key. Such changes to the environment affect subprocesses started with os.system(), popen() or fork() and execv().

When unsetenv() is supported, deletion of items in os.environ is automatically translated into a corresponding call to unsetenv(); however, calls to unsetenv() don’t update os.environ, so it is actually preferable to delete items of os.environ.

Availability: most flavors of Unix, Windows.

File Object Creation

This function creates new file objects. (See also open() for opening file descriptors.)

**os.fdopen(fd, \*args, \*\*kwargs)**

Return an open file object connected to the file descriptor fd. This is an alias of the open() built-in function and accepts the same arguments. The only difference is that the first argument of fdopen() must always be an integer.

File Descriptor Operations

These functions operate on I/O streams referenced using file descriptors.

File descriptors are small integers corresponding to a file that has been opened by the current process. For example, standard input is usually file descriptor 0, standard output is 1, and standard error is 2. Further files opened by a process will then be assigned 3, 4, 5, and so forth. The name “file descriptor” is slightly deceptive; on Unix platforms, sockets and pipes are also referenced by file descriptors.

The fileno() method can be used to obtain the file descriptor associated with a file object when required. Note that using the file descriptor directly will bypass the file object methods, ignoring aspects such as internal buffering of data.

**os.close(fd)**

Close file descriptor fd.

Note This function is intended for low-level I/O and must be applied to a file descriptor as returned by os.open() or pipe(). To close a “file object” returned by the built-in function open() or by popen() or fdopen(), use its close() method.

**os.closerange(fd\_low, fd\_high)**

Close all file descriptors from fd\_low (inclusive) to fd\_high (exclusive), ignoring errors. Equivalent to (but much faster than):

for fd in range(fd\_low, fd\_high):

try:

os.close(fd)

except OSError:

pass

os.device\_encoding(fd)

Return a string describing the encoding of the device associated with fd if it is connected to a terminal; else return None.

**os.dup(fd)**

Return a duplicate of file descriptor fd. The new file descriptor is non-inheritable.

On Windows, when duplicating a standard stream (0: stdin, 1: stdout, 2: stderr), the new file descriptor is inheritable.

Changed in version 3.4: The new file descriptor is now non-inheritable.

**os.dup2(fd, fd2, inheritable=True)**

Duplicate file descriptor fd to fd2, closing the latter first if necessary. Return fd2. The new file descriptor is inheritable by default or non-inheritable if inheritable is False.

Changed in version 3.4: Add the optional inheritable parameter.

Changed in version 3.7: Return fd2 on success. Previously, None was always returned.

**os.fchmod(fd, mode)**

Change the mode of the file given by fd to the numeric mode. See the docs for chmod() for possible values of mode. As of Python 3.3, this is equivalent to os.chmod(fd, mode).

Availability: Unix.

**os.fchown(fd, uid, gid)**

Change the owner and group id of the file given by fd to the numeric uid and gid. To leave one of the ids unchanged, set it to -1. See chown(). As of Python 3.3, this is equivalent to os.chown(fd, uid, gid).

Availability: Unix.

**os.fdatasync(fd)**

Force write of file with filedescriptor fd to disk. Does not force update of metadata.

Availability: Unix.

Note: This function is not available on MacOS.

**os.fpathconf(fd, name)**

Return system configuration information relevant to an open file. name specifies the configuration value to retrieve; it may be a string which is the name of a defined system value; these names are specified in a number of standards (POSIX.1, Unix 95, Unix 98, and others). Some platforms define additional names as well. The names known to the host operating system are given in the pathconf\_names dictionary. For configuration variables not included in that mapping, passing an integer for name is also accepted.

If name is a string and is not known, ValueError is raised. If a specific value for name is not supported by the host system, even if it is included in pathconf\_names, an OSError is raised with errno.EINVAL for the error number.

As of Python 3.3, this is equivalent to os.pathconf(fd, name).

Availability: Unix.

**os.fstat(fd)**

Get the status of the file descriptor fd. Return a stat\_result object.

As of Python 3.3, this is equivalent to os.stat(fd).

See also The stat() function.

**os.fstatvfs(fd)**

Return information about the filesystem containing the file associated with file descriptor fd, like statvfs(). As of Python 3.3, this is equivalent to os.statvfs(fd).

Availability: Unix.

**os.fsync(fd)**

Force write of file with filedescriptor fd to disk. On Unix, this calls the native fsync() function; on Windows, the MS \_commit() function.

If you’re starting with a buffered Python file object f, first do f.flush(), and then do os.fsync(f.fileno()), to ensure that all internal buffers associated with f are written to disk.

Availability: Unix, Windows.

**os.ftruncate(fd, length)**

Truncate the file corresponding to file descriptor fd, so that it is at most length bytes in size. As of Python 3.3, this is equivalent to os.truncate(fd, length).

Availability: Unix, Windows.

Changed in version 3.5: Added support for Windows

**os.get\_blocking(fd)**

Get the blocking mode of the file descriptor: False if the O\_NONBLOCK flag is set, True if the flag is cleared.

See also set\_blocking() and socket.socket.setblocking().

Availability: Unix.

New in version 3.5.

**os.isatty(fd)**

Return True if the file descriptor fd is open and connected to a tty(-like) device, else False.

**os.lockf(fd, cmd, len)**

Apply, test or remove a POSIX lock on an open file descriptor. fd is an open file descriptor. cmd specifies the command to use - one of F\_LOCK, F\_TLOCK, F\_ULOCK or F\_TEST. len specifies the section of the file to lock.

Availability: Unix.

New in version 3.3.

os.F\_LOCK

os.F\_TLOCK

os.F\_ULOCK

os.F\_TEST

Flags that specify what action lockf() will take.

Availability: Unix.

New in version 3.3.

**os.lseek(fd, pos, how)**

Set the current position of file descriptor fd to position pos, modified by how: SEEK\_SET or 0 to set the position relative to the beginning of the file; SEEK\_CUR or 1 to set it relative to the current position; SEEK\_END or 2 to set it relative to the end of the file. Return the new cursor position in bytes, starting from the beginning.

os.SEEK\_SET

os.SEEK\_CUR

os.SEEK\_END

Parameters to the lseek() function. Their values are 0, 1, and 2, respectively.

New in version 3.3: Some operating systems could support additional values, like os.SEEK\_HOLE or os.SEEK\_DATA.

**os.open(path, flags, mode=0o777, \*, dir\_fd=None)**

Open the file path and set various flags according to flags and possibly its mode according to mode. When computing mode, the current umask value is first masked out. Return the file descriptor for the newly opened file. The new file descriptor is non-inheritable.

For a description of the flag and mode values, see the C run-time documentation; flag constants (like O\_RDONLY and O\_WRONLY) are defined in the os module. In particular, on Windows adding O\_BINARY is needed to open files in binary mode.

This function can support paths relative to directory descriptors with the dir\_fd parameter.

Changed in version 3.4: The new file descriptor is now non-inheritable.

Note This function is intended for low-level I/O. For normal usage, use the built-in function open(), which returns a file object with read() and write() methods (and many more). To wrap a file descriptor in a file object, use fdopen().

New in version 3.3: The dir\_fd argument.

Changed in version 3.5: If the system call is interrupted and the signal handler does not raise an exception, the function now retries the system call instead of raising an InterruptedError exception (see PEP 475 for the rationale).

Changed in version 3.6: Accepts a path-like object.

The following constants are options for the flags parameter to the open() function. They can be combined using the bitwise OR operator |. Some of them are not available on all platforms. For descriptions of their availability and use, consult the open(2) manual page on Unix or the MSDN on Windows.

os.O\_RDONLY

os.O\_WRONLY

os.O\_RDWR

os.O\_APPEND

os.O\_CREAT

os.O\_EXCL

os.O\_TRUNC

The above constants are available on Unix and Windows.

os.O\_DSYNC

os.O\_RSYNC

os.O\_SYNC

os.O\_NDELAY

os.O\_NONBLOCK

os.O\_NOCTTY

os.O\_CLOEXEC

The above constants are only available on Unix.

Changed in version 3.3: Add O\_CLOEXEC constant.

os.O\_BINARY

os.O\_NOINHERIT

os.O\_SHORT\_LIVED

os.O\_TEMPORARY

os.O\_RANDOM

os.O\_SEQUENTIAL

os.O\_TEXT

The above constants are only available on Windows.

os.O\_ASYNC

os.O\_DIRECT

os.O\_DIRECTORY

os.O\_NOFOLLOW

os.O\_NOATIME

os.O\_PATH

os.O\_TMPFILE

os.O\_SHLOCK

os.O\_EXLOCK

The above constants are extensions and not present if they are not defined by the C library.

Changed in version 3.4: Add O\_PATH on systems that support it. Add O\_TMPFILE, only available on Linux Kernel 3.11 or newer.

os.openpty()

Open a new pseudo-terminal pair. Return a pair of file descriptors (master, slave) for the pty and the tty, respectively. The new file descriptors are non-inheritable. For a (slightly) more portable approach, use the pty module.

Availability: some flavors of Unix.

Changed in version 3.4: The new file descriptors are now non-inheritable.

os.pipe()

Create a pipe. Return a pair of file descriptors (r, w) usable for reading and writing, respectively. The new file descriptor is non-inheritable.

Availability: Unix, Windows.

Changed in version 3.4: The new file descriptors are now non-inheritable.

os.pipe2(flags)

Create a pipe with flags set atomically. flags can be constructed by ORing together one or more of these values: O\_NONBLOCK, O\_CLOEXEC. Return a pair of file descriptors (r, w) usable for reading and writing, respectively.

Availability: some flavors of Unix.

New in version 3.3.

os.posix\_fallocate(fd, offset, len)

Ensures that enough disk space is allocated for the file specified by fd starting from offset and continuing for len bytes.

Availability: Unix.

New in version 3.3.

os.posix\_fadvise(fd, offset, len, advice)

Announces an intention to access data in a specific pattern thus allowing the kernel to make optimizations. The advice applies to the region of the file specified by fd starting at offset and continuing for len bytes. advice is one of POSIX\_FADV\_NORMAL, POSIX\_FADV\_SEQUENTIAL, POSIX\_FADV\_RANDOM, POSIX\_FADV\_NOREUSE, POSIX\_FADV\_WILLNEED or POSIX\_FADV\_DONTNEED.

Availability: Unix.

New in version 3.3.

os.POSIX\_FADV\_NORMAL

os.POSIX\_FADV\_SEQUENTIAL

os.POSIX\_FADV\_RANDOM

os.POSIX\_FADV\_NOREUSE

os.POSIX\_FADV\_WILLNEED

os.POSIX\_FADV\_DONTNEED

Flags that can be used in advice in posix\_fadvise() that specify the access pattern that is likely to be used.

Availability: Unix.

New in version 3.3.

os.pread(fd, n, offset)

Read at most n bytes from file descriptor fd at a position of offset, leaving the file offset unchanged.

Return a bytestring containing the bytes read. If the end of the file referred to by fd has been reached, an empty bytes object is returned.

Availability: Unix.

New in version 3.3.

os.preadv(fd, buffers, offset, flags=0)

Read from a file descriptor fd at a position of offset into mutable bytes-like objects buffers, leaving the file offset unchanged. Transfer data into each buffer until it is full and then move on to the next buffer in the sequence to hold the rest of the data.

The flags argument contains a bitwise OR of zero or more of the following flags:

RWF\_HIPRI

RWF\_NOWAIT

Return the total number of bytes actually read which can be less than the total capacity of all the objects.

The operating system may set a limit (sysconf() value 'SC\_IOV\_MAX') on the number of buffers that can be used.

Combine the functionality of os.readv() and os.pread().

Availability: Linux 2.6.30 and newer, FreeBSD 6.0 and newer, OpenBSD 2.7 and newer. Using flags requires Linux 4.6 or newer.

# **Numpy**



NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier.

An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become Numeric,[4] also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at Massachusetts Institute of Technology (MIT), joined the Corporation for National Research Initiatives (CNRI) to work on JPython in 1997 leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

A new package called Numarray was written as a more flexible replacement for Numeric. Like Numeric, it is now deprecated. Numarray had faster operations for large arrays, but was slower than Numeric on small ones, so for a time both packages were used for different use cases. The last version of Numeric v24.2 was released on 11 November 2005 and numarray v1.5.2 was released on 24 August 2006.

There was a desire to get Numeric into the Python standard library, but Guido van Rossum decided that the code was not maintainable in its state then.

In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006.[7] This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0.[13]

In 2011, PyPy started development on an implementation of the NumPy API for PyPy. It is not yet fully compatible with NumPy.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[16] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

**Traits**

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[16] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

The ndarray data structure

The core functionality of NumPy is its "ndarray", for n-dimensional array, data structure. These arrays are strided views on memory.[7] In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type.

Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory-mapped ndarrays.[7]

Limitations

Inserting or appending entries to an array is not as trivially possible as it is with Python's lists. The np.pad(...) routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it. NumPy's np.concatenate([a1,a2]) operation does not actually link the two arrays but returns a new one, filled with the entries from both given arrays in sequence. Reshaping the dimensionality of an array with np.reshape(...) is only possible as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome this limitation.[17]

Algorithms that are not expressible as a vectorized operation will typically run slowly because they must be implemented in "pure Python", while vectorization may increase memory complexity of some operations from constant to linear, because temporary arrays must be created that are as large as the inputs. Runtime compilation of numerical code has been implemented by several groups to avoid these problems; open source solutions that interoperate with NumPy

include scipy.weave, numexpr[18] and Numba.[19] Cython and Pythran are static-compiling alternatives to these.

**The Basics**

NumPy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called axes.

For example, the coordinates of a point in 3D space [1, 2, 1] has one axis. That axis has 3 elements in it, so we say it has a length of 3. In the example pictured below, the array has 2 axes. The first axis has a length of 2, the second axis has a length of 3.

[[ 1., 0., 0.],

[ 0., 1., 2.]]

NumPy’s array class is called ndarray. It is also known by the alias array. Note that numpy.array is not the same as the Standard Python Library class array.array, which only handles one-dimensional arrays and offers less functionality. The more important attributes of an ndarray object are:

**ndarray.ndim**

the number of axes (dimensions) of the array.

**ndarray.shape**

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with nrows and m columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

**ndarray.size**

the total number of elements of the array. This is equal to the product of the elements of shape.

**ndarray.dtype**

an object describing the type of the elements in the array. One can create or specify dtype’s using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

**ndarray.itemsize**

the size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize.

**ndarray.data**

the buffer containing the actual elements of the array. Normally, we won’t need to use this attribute because we will access the elements in an array using indexing facilities.

An example

>>>

>>> import numpy as np

>>> a = np.arange(15).reshape(3, 5)

>>> a

array([[ 0, 1, 2, 3, 4],

[ 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14]])

>>> a.shape

(3, 5)

>>> a.ndim

2

>>> a.dtype.name

'int64'

>>> a.itemsize

8

>>> a.size

15

>>> type(a)

<type 'numpy.ndarray'>

>>> b = np.array([6, 7, 8])

>>> b

array([6, 7, 8])

>>> type(b)

<type 'numpy.ndarray'>

**Array Creation**

There are several ways to create arrays.

For example, you can create an array from a regular Python list or tuple using the array function. The type of the resulting array is deduced from the type of the elements in the sequences.

>>>

>>> import numpy as np

>>> a = np.array([2,3,4])

>>> a

array([2, 3, 4])

>>> a.dtype

dtype('int64')

>>> b = np.array([1.2, 3.5, 5.1])

>>> b.dtype

dtype('float64')

A frequent error consists in calling array with multiple numeric arguments, rather than providing a single list of numbers as an argument.

>>>

>>> a = np.array(1,2,3,4) # WRONG

>>> a = np.array([1,2,3,4]) # RIGHT

array transforms sequences of sequences into two-dimensional arrays, sequences of sequences of sequences into three-dimensional arrays, and so on.

>>>

>>> b = np.array([(1.5,2,3), (4,5,6)])

>>> b

array([[ 1.5, 2. , 3. ],

[ 4. , 5. , 6. ]])

The type of the array can also be explicitly specified at creation time:

>>>

>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )

>>> c

array([[ 1.+0.j, 2.+0.j],

[ 3.+0.j, 4.+0.j]])

Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with initial placeholder content. These minimize the necessity of growing arrays, an expensive operation.

The function zeros creates an array full of zeros, the function ones creates an array full of ones, and the function empty creates an array whose initial content is random and depends on the state of the memory. By default, the dtype of the created array isfloat64.

>>>

>>> np.zeros( (3,4) )

array([[ 0., 0., 0., 0.],

[ 0., 0., 0., 0.],

[ 0., 0., 0., 0.]])

>>> np.ones( (2,3,4), dtype=np.int16 ) # dtype can also be specified

array([[[ 1, 1, 1, 1],

[ 1, 1, 1, 1],

[ 1, 1, 1, 1]],

[[ 1, 1, 1, 1],

[ 1, 1, 1, 1],

[ 1, 1, 1, 1]]], dtype=int16)

>>> np.empty( (2,3) ) # uninitialized, output may vary

array([[ 3.73603959e-262, 6.02658058e-154, 6.55490914e-260],

[ 5.30498948e-313, 3.14673309e-307, 1.00000000e+000]])

To create sequences of numbers, NumPy provides a function analogous to range that returns arrays instead of lists.

>>>

>>> np.arange( 10, 30, 5 )

array([10, 15, 20, 25])

>>> np.arange( 0, 2, 0.3 ) # it accepts float arguments

array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])

When arange is used with floating point arguments, it is generally not possible to predict the number of elements obtained, due to the finite floating point precision. For this reason, it is usually better to use the function linspace that receives as an argument the number of elements that we want, instead of the step:

>>>

>>> from numpy import pi

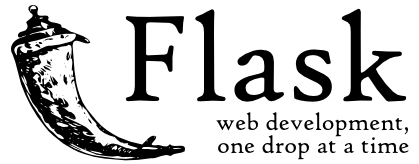
>>> np.linspace( 0, 2, 9 ) # 9 numbers from 0 to 2

array([ 0. , 0.25, 0.5 , 0.75, 1. , 1.25, 1.5 , 1.75, 2. ])

>>> x = np.linspace( 0, 2\*pi, 100 ) # useful to evaluate function at lots of points

>>> f = np.sin(x)

Flask



[**Star**](https://github.com/pallets/flask)[**53,671**](https://github.com/pallets/flask/stargazers)

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around [Werkzeug](https://palletsprojects.com/p/werkzeug) and [Jinja](https://palletsprojects.com/p/jinja) and has become one of the most popular Python web application frameworks.

Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy.

**from** flask **import** Flask**,** escape**,** request

app **=** Flask**(**\_\_name\_\_**)**

**@app.**route**(**'/'**)**

**def** hello**():**

name **=** request**.**args**.**get**(**"name"**,** "World"**)**

**return** f'Hello, {escape(name)}!'

$ env FLASK\_APP=hello.py flask run

\* Serving Flask app "hello"

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Applications that use the Flask framework include Pinterest and LinkedIn.

HTML5

HTML5 is a markup language used for structuring and presenting content on the World Wide Web. It is the fifth and last major HTML version that is a World Wide Web Consortium (W3C) recommendation. The current specification is known as the HTML Living Standard. It is maintained by a consortium of the major browser vendors (Apple, Google, Mozilla, and Microsoft), the Web Hypertext Application Technology Working Group (WHATWG).



HTML5 was first released in a public-facing form on 22 January 2008, with a major update and "W3C Recommendation" status in October 2014.[2][5] Its goals were to improve the language with support for the latest multimedia and other new features; to keep the language both easily readable by humans and consistently understood by computers and devices such as web browsers, parsers, etc., without XHTML's rigidity; and to remain backward-compatible with older software. HTML5 is intended to subsume not only HTML 4 but also XHTML 1 and DOM Level 2 HTML.

HTML5 includes detailed processing models to encourage more interoperable implementations; it extends, improves, and rationalizes the markup available for documents and introduces markup and application programming interfaces (APIs) for complex web applications. For the same reasons, HTML5 is also a candidate for cross-platform mobile applications because it includes features designed with low-powered devices in mind.

Many new syntactic features are included. To natively include and handle multimedia and graphical content, the new <video>, <audio> and <canvas> elements were added, and support for scalable vector graphics (SVG) content and MathML for mathematical formulas was also added. To enrich the semantic content of documents, new page structure elements such as <main>, <section>, <article>, <header>, <footer>, <aside>, <nav>, and <figure> are added. New attributes were introduced, some elements and attributes were removed, and others such as <a>, <cite>, and <menu> were changed, redefined, or standardized. The APIs and Document Object Model (DOM) are now fundamental parts of the HTML5 specification, and HTML5 also better defines the processing for any invalid documents.

**CSS(Cascading Style Sheet)**

**CSS** is used to control the style of a web document in a simple and easy way.

**CSS** is the acronym for **"Cascading Style Sheet"**. This tutorial covers both the versions CSS1,CSS2 and CSS3, and gives a complete understanding of CSS, starting from its basics to advanced concepts.

Why to Learn CSS?

**Cascading Style Sheets**, fondly referred to as **CSS**, is a simple design language intended to simplify the process of making web pages presentable.

**CSS** is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning CSS:

* **Create Stunning Web site** - CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs,variations in display for different devices and screen sizes as well as a variety of other effects.
* **Become a web designer** - If you want to start a carrer as a professional web designer, HTML and CSS designing is a must skill.
* **Control web** - CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.
* **Learn other languages** - Once you understands the basic of HTML and CSS then other related technologies like javascript, php, or angular are become easier to understand.

Hello World using CSS.

Just to give you a little excitement about CSS, I'm going to give you a small conventional CSS Hello World program, You can try it using Demo link.

<!DOCTYPE html>

<html>

<head>

<title>This is document title</title>

<style>

h1 {

color: #36CFFF;

}

</style>

</head>

<body>

<h1>Hello World!</h1>

</body>

</html>

Applications of CSS

As mentioned before, CSS is one of the most widely used style language over the web. I'm going to list few of them here:

* **CSS saves time** - You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.
* **Pages load faster** - If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
* **Easy maintenance** - To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
* **Superior styles to HTML** - CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
* **Multiple Device Compatibility** - Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
* **Global web standards**- Now HTML attributes are being deprecated and it is being recommended to use CSS. So its a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

Prerequisites

You should be familiar with:

* Basic word processing using any text editor.
* How to create directories and files.
* How to navigate through different directories.
* Internet browsing using popular browsers like Internet Explorer or Firefox.
* Developing simple Web Pages using HTML or XHTML.

**Run Time Output Screenshots: (Add Output Screenshots here)**

**Conclusion: (Add few lines of Conclusion as per your understanding of Project)**

The System was made and Tested for the output results.

WE WISH YOU ALL THE BEST FOR YOUR CAREER