

Hello.

JavaScript for Pythoners (and vice versa)

Nov 29, 2023.

© Huge 2021



About me.

Cami Gómez.

@camigomezdev



About me.

- 1. Python and JavaScript.**
- 2. Installation.**
- 3. REPL.**
- 4. Syntax.**
- 5. List comprehension and functional programming.**
- 6. Pattern matching.**
- 7. Lambda functions.**
- 8. Type hints and Typescript.**

- 1. Python and JavaScript.**
- 2. Installation.**
- 3. REPL.**
- 4. Syntax.**
- 5. List comprehension and functional programming.**
- 6. Pattern matching.**
- 7. Lambda functions.**
- 8. Type hints and Typescript.**

What's Python.

Interpreted, object-oriented, high-level programming language with **dynamic** semantics. Python's design philosophy emphasizes code readability.

What's JavaScript.

Javascript is a **high-level**, often **just-in-time compiled**, and **multi-paradigm** programming language. It has **curly-bracket** syntax, **dynamic** typing, **prototype-based** object-orientation.

- 
- 1. Python and JavaScript.**
 - 2. Installation.**
 - 3. REPL.**
 - 4. Syntax.**
 - 5. List comprehension and functional programming.**
 - 6. Pattern matching.**
 - 7. Lambda functions.**
 - 8. Type hints and Typescript.**

Installation.

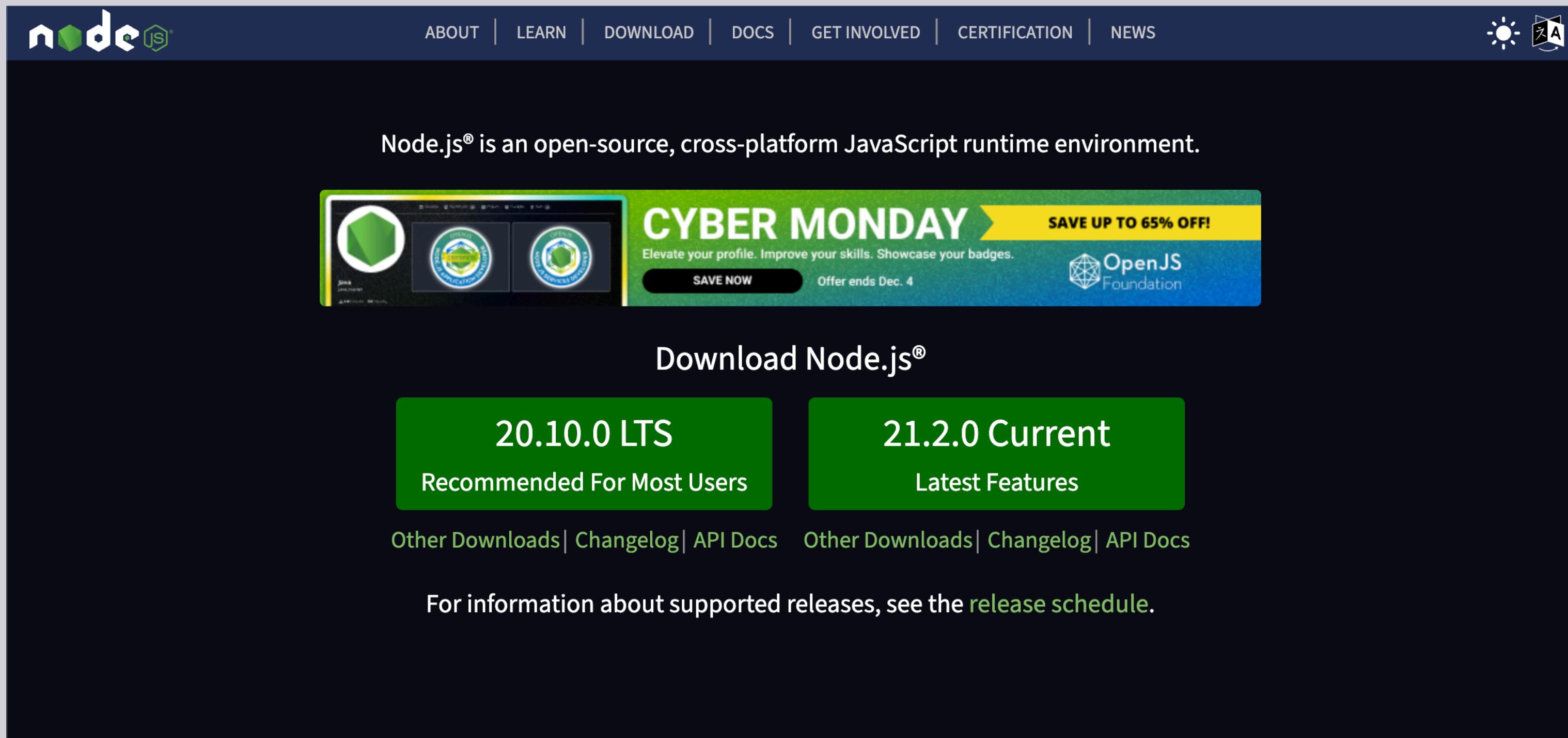
Python.

The screenshot shows the Python.org homepage. At the top, there's a navigation bar with links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the navigation is the Python logo and a search bar with buttons for 'Donate', 'Search', 'GO', and 'Socialize'. A secondary navigation bar below the main one has links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area features a large yellow call-to-action button labeled 'Download Python 3.9.6'. To the left of this button, text says 'Download the latest version for Mac OS X'. Below this, there are links for other operating systems: 'Windows', 'Linux/UNIX', 'Mac OS X', and 'Other'. Further down, there are links for 'Prereleases' and 'Docker images'. At the bottom left, it says 'Looking for Python 2.7? See below for specific releases'. The background of the main content area features a cartoon illustration of two boxes descending from the sky on parachutes.

<https://www.python.org/downloads/>

Installation.

JavaScript.

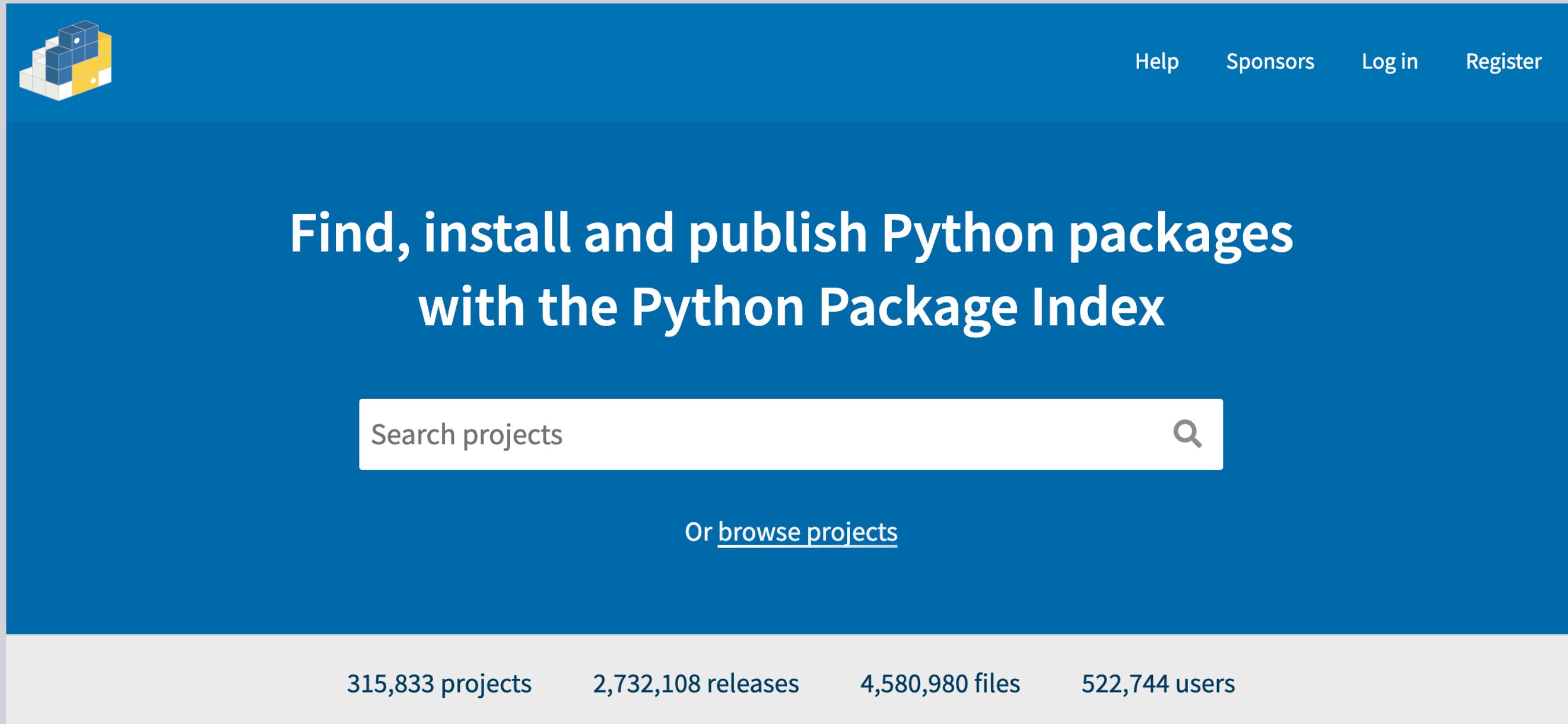


The screenshot shows the official Node.js website. At the top, there's a dark blue header bar with the Node.js logo on the left and navigation links for ABOUT, LEARN, DOWNLOAD, DOCS, GET INVOLVED, CERTIFICATION, and NEWS. On the right side of the header are icons for a sun, a gear, and a person. Below the header, a large white text area contains the sentence: "Node.js® is an open-source, cross-platform JavaScript runtime environment." To the left of this text is a small image showing three circular badges related to Node.js. To the right is a promotional banner for "CYBER MONDAY" with the text "SAVE UP TO 65% OFF!" and "Elevate your profile. Improve your skills. Showcase your badges." It includes a "SAVE NOW" button and the note "Offer ends Dec. 4". Below this, there's a section titled "Download Node.js®" with two main download options: "20.10.0 LTS Recommended For Most Users" and "21.2.0 Current Latest Features". At the bottom of this section, there are links for "Other Downloads | Changelog | API Docs" for each release. A note at the very bottom says: "For information about supported releases, see the [release schedule](#)".

<https://nodejs.org/en/>

[Installation.](#)

PyPI and Pip.

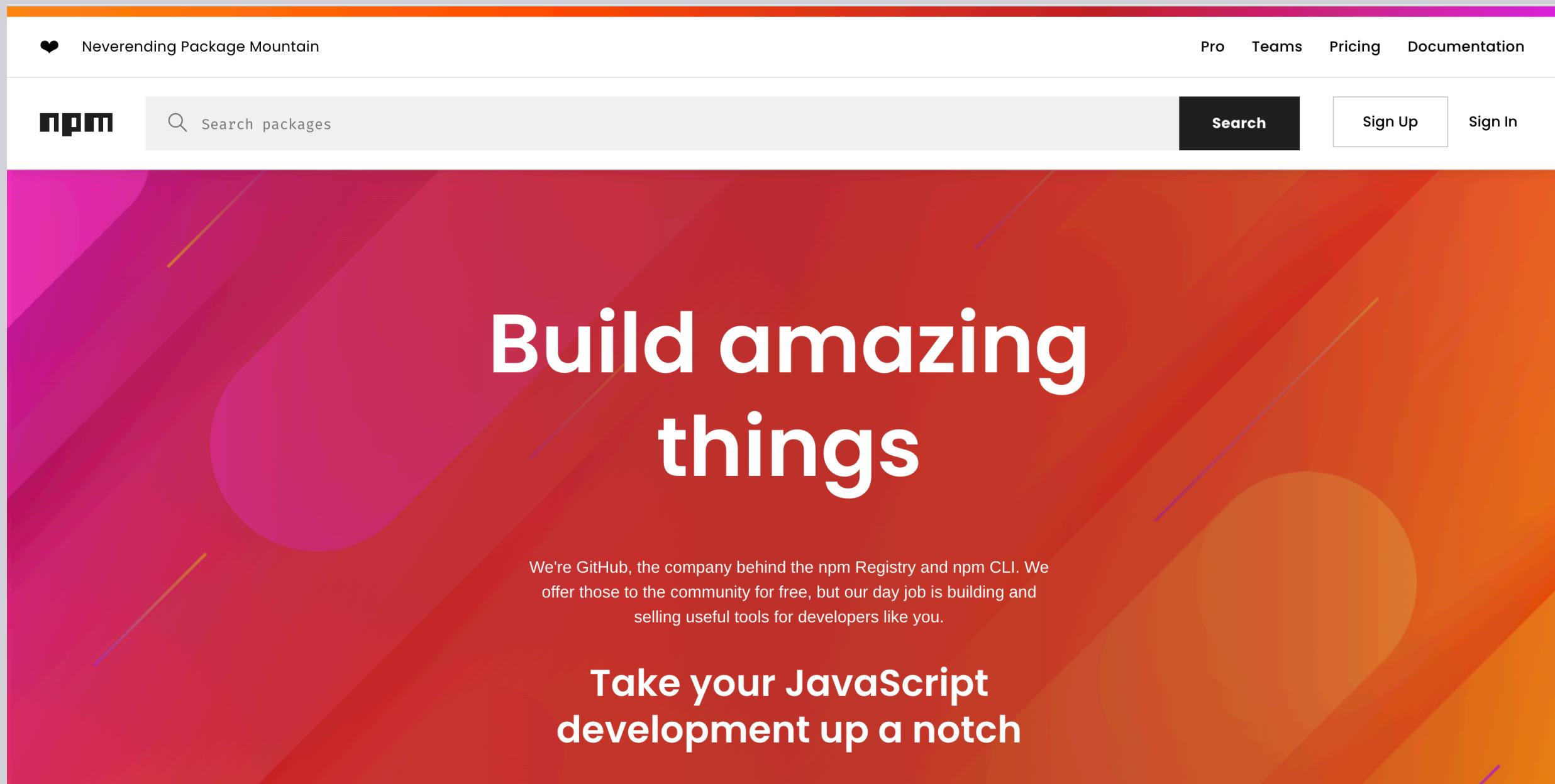


The screenshot shows the PyPI homepage. At the top, there's a navigation bar with links for Help, Sponsors, Log in, and Register. To the left of the main content area is a small icon of three stacked cubes in white, yellow, and blue. The main heading "Find, install and publish Python packages with the Python Package Index" is centered in large white text on a blue background. Below this is a search bar with the placeholder "Search projects" and a magnifying glass icon. Underneath the search bar is a link "Or [browse projects](#)". At the bottom of the page, a white footer bar displays statistics: "315,833 projects", "2,732,108 releases", "4,580,980 files", and "522,744 users".

pip is the package installer for Python. You can use it to install packages from the Python Package Index and other indexes.

Installation.

NPM.

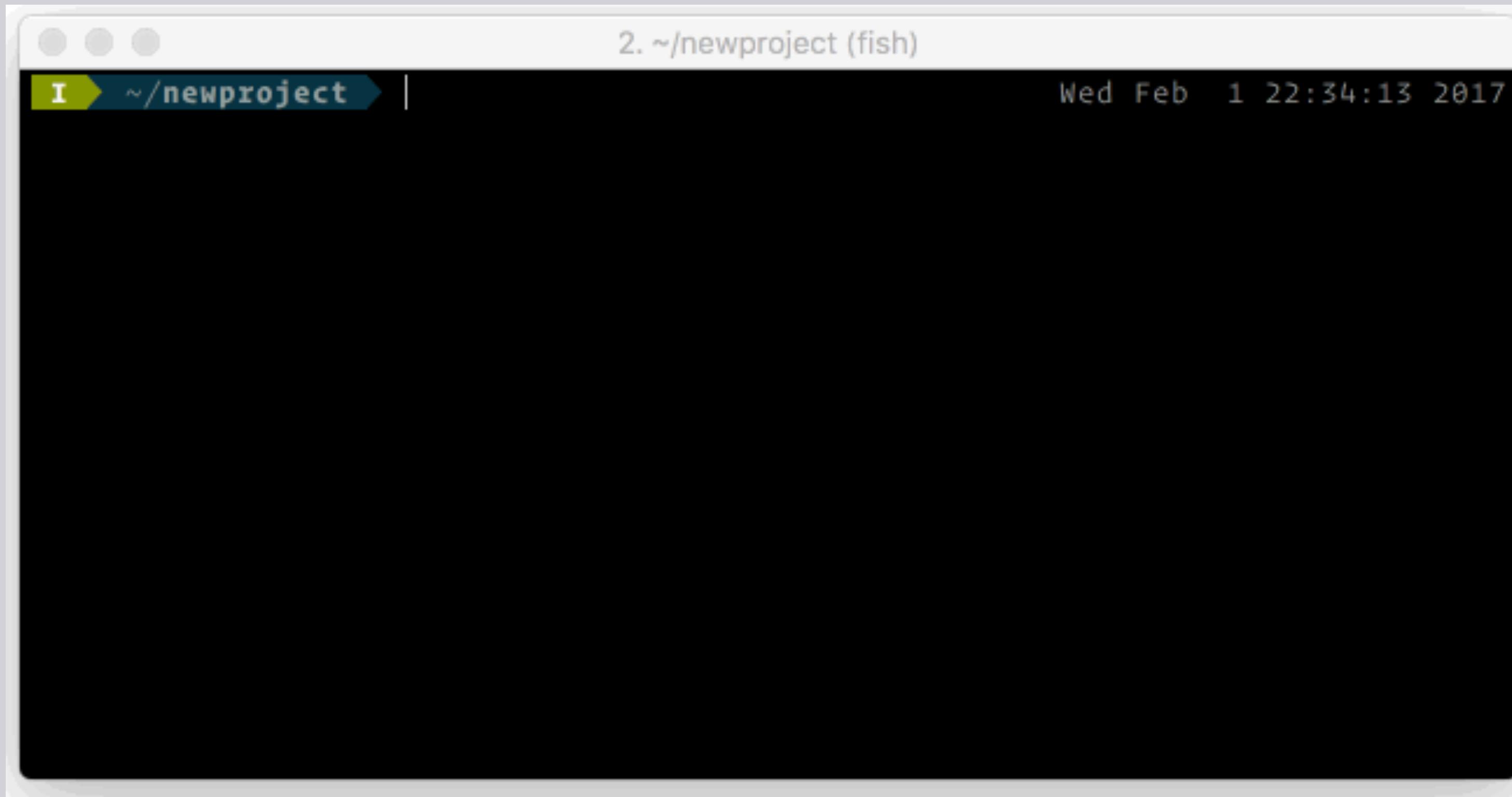


npm is a package manager for the JavaScript programming language.

npm is the default package manager for the JavaScript runtime

Installation.

Pipenv: Python Dev Workflow for Humans.



Brings the best of all packaging worlds (bundler, composer, npm, cargo, yarn, etc.) to the Python world.

NVM.



The screenshot shows a dark-themed terminal window with white text. At the top, the title "Intro" is visible. Below it, a note states: "nvm allows you to quickly install and use different versions of node via the command line." Under the heading "Example:", several commands are shown: "\$ nvm use 16", "Now using node v16.9.1 (npm v7.21.1)", "\$ node -v", "v16.9.1", "\$ nvm use 14", "Now using node v14.18.0 (npm v6.14.15)", "\$ node -v", "v14.18.0", "\$ nvm install 12", "Now using node v12.22.6 (npm v6.14.5)", "\$ node -v", "v12.22.6". At the bottom of the terminal window, the text "Simple as that!" is displayed.

Node Version Manager.

Installation.

Virtualenv.

The screenshot shows the PyPI project page for 'virtualenv'. At the top, there's a search bar with 'Search projects' and a magnifying glass icon. To the right are links for 'Help', 'Sponsors', 'Log in', and 'Register'. Below the header, the package name 'virtualenv 20.4.7' is displayed in large white text on a blue background. To the right of the version number is a green button with a checkmark and the text 'Latest version'. On the left, there's a code block containing the command 'pip install virtualenv' followed by a file icon. To the right of the command is the release date 'Released: May 24, 2021'. Below the main header, the text 'Virtual Python Environment builder' is visible.

virtualenv 20.4.7

pip install virtualenv

Released: May 24, 2021

Virtual Python Environment builder

Creates isolated Python environments.

Installation.

Python.

PyPI.

Pip.

Virtualenv.

Pipenv.

JavaScript.

npm (site).

npm (command).

nvm.

Node project with nvm.

- 1. Python and JavaScript.**
- 2. Installation.**
- 3. REPL.**
- 4. Syntax.**
- 5. List comprehension and functional programming.**
- 6. Pattern matching.**
- 7. Lambda functions.**
- 8. Type hints and Typescript.**



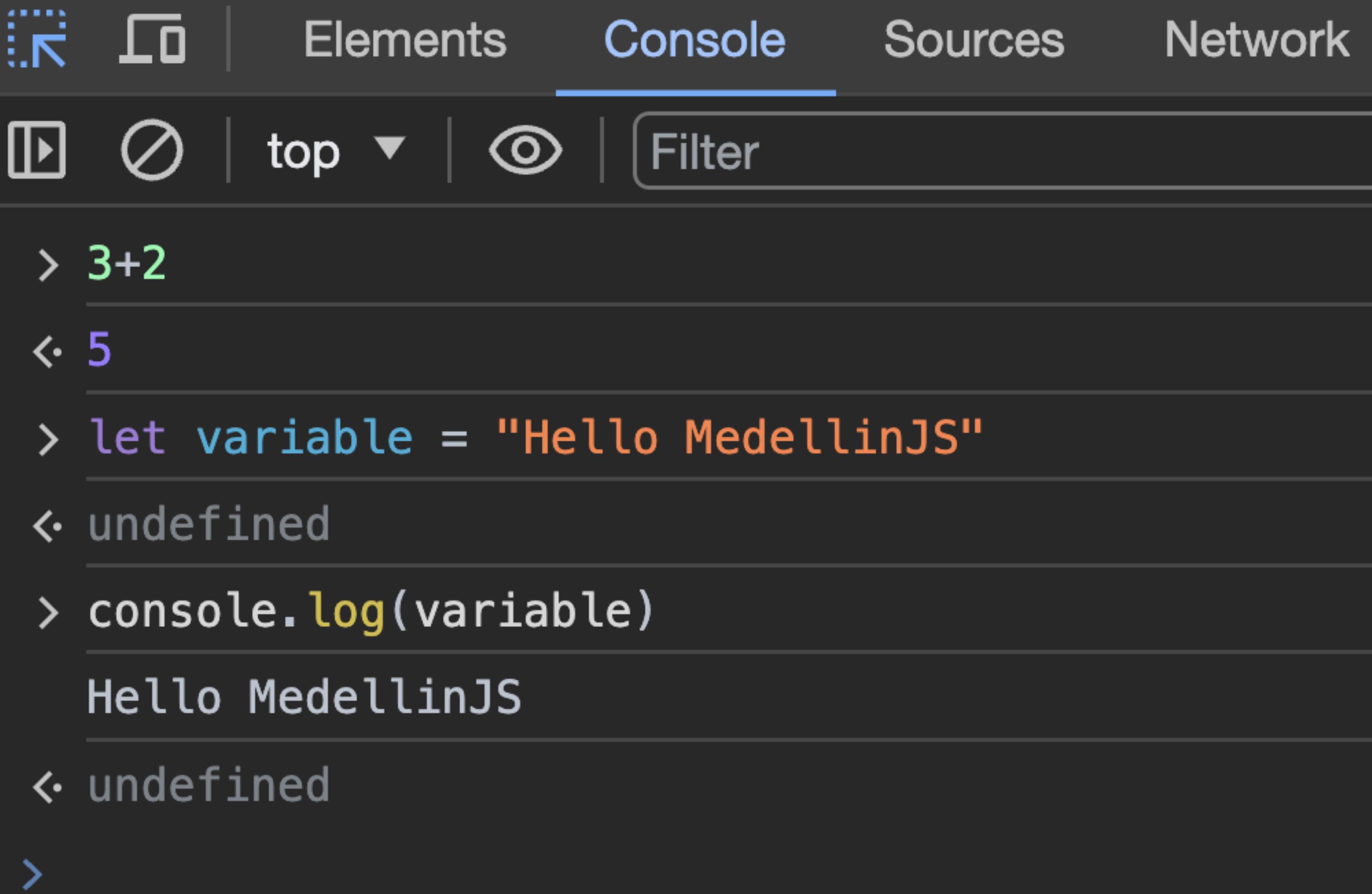
Read, Evaluate, Print, Loop.

Unlike running a file containing Python code, in the REPL you can type commands and instantly see the output printed out.

```
[H12428:~ $ python3
Python 3.11.5 (main, Aug 24 2023, 15:18:16) [Clang 14
.0.3 (clang-1403.0.22.14.1)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
[>>> 3+2
5
[>>> variable = "Hello MedellinJS"
[>>> print(variable)
Hello MedellinJS
>>> |
```

REPL.

Interactive Console in the Browser.



The screenshot shows the 'Console' tab of a browser's developer tools. At the top, there are icons for copy, paste, and clear, followed by tabs for 'Elements', 'Console' (which is selected), 'Sources', and 'Network'. Below the tabs are controls for play/pause, stop, and zoom level (top, bottom). A 'Filter' input field is also present. The main area displays the following interaction:

```
> 3+2
<- 5
> let variable = "Hello MedellinJS"
<- undefined
> console.log(variable)
Hello MedellinJS
<- undefined
>
```

```
[H12428:~ $ node
Welcome to Node.js v18.9.0.
Type ".help" for more information.
[> 3+2
5
[> let variable = "Hello MedellinJS"
undefined
[> console.log(variable)
Hello MedellinJS
undefined
> |
```

Node.js REPL.

- 
- 1. Python and JavaScript.**
 - 2. Installation.**
 - 3. REPL.**
 - 4. Syntax.**
 - 5. List comprehension and functional programming.**
 - 6. Pattern matching.**
 - 7. Lambda functions.**
 - 8. Type hints and Typescript.**

Syntax.



```
def my_function():
    if True:
        print("It works!")

my_function()
# It works!
```

Python.



```
function myFunction() {
    if (true) {
        console.log("It works!")
    }
}

myFunction()
// It works!
```

JavaScript.

Operator.	Python.	JavaScript.
Addition.	+	+
Subtraction.	-	-
Multiplication.	*	*
Exponent.	**	**
Division.	/	/
Floor division.	//	N/A
Modulo.	%	%
Increment.	+=	++/+=
Decrement.	-=	--/-=

Operator.	Python.	JavaScript.
Greater than.	>	>
Less than.	<	<
Greater or equal than.	>=	>=
Less or equal than.	<=	<=
Equal.	==	==
Strict (triple) equal.	N/A	====
Not equal.	!=	!=
Strict not equal.	N/A	!==

Logical operators.

Operator.	Python.	JavaScript.
And.	and	&&
Or.	or	
Negation.	not	!

Data types.

Python.	JavaScript.
float.	Number.
int.	BigInt.
int.	Number.
string.	String.
boolean.	Boolean.
None.	Null.
N/a.	Undefined.

Data types.

Python.	JavaScript.
list.	Array.
dictionary.	Object.
set.	Set.
tuple.	N/a.

Variables.



```
variable = "This is a variable"  
CONSTANT = "This is a constant"
```

Python.



```
var variableOne = "This is a variable"  
let variableTwo = "This is another variable"  
const CONSTANT = "This is a constant"
```

JavaScript.

Conditionals.



```
n_cats = 10

if n_cats < 3:
    print("So sad")
elif n_cats >= 3 and n_cats <= 5:
    print("Nice!")
else:
    print("You are a lucky person")

# You are a lucky person
```



```
let n_cats = 10

if (n_cats < 3) {
    console.log("So sad")
}
else if (n_cats >= 3 && n_cats <= 5) {
    console.log("Nice!")
}
else {
    console.log("You are a lucky person!")
}

// You are a lucky person!
```

Python.

JavaScript.

Conditionals.



```
n_cats = 3
message = "Nice!" if n_cats >= 3 else "So sad"
print(message)
# Nice!
```



```
let n_cats = 3
let message = n_cats >= 3 ? "Nice!" : "So sad"
console.log(message)
// Nice!
```

Python.

JavaScript.

For loop.



```
for x in range(6):  
    print(x)
```

0,1,2,3,4,5



```
for (let x = 0; x < 6; x++) {  
    console.log(x)  
}  
// 0,1,2,3,4,5
```

Python.

For loop.

JavaScript.

For, for/of, for/in.

For loop.

```
my_list = [1, 2, 3, 4]
for value in my_list:
    print(value)

# 1,2,3,4
```

Python.

For loop.



```
myArray = [1, 2, 3, 4]
for (value of myArray) {
    console.log(value)
}

// 1,2,3,4
```

JavaScript.

For, for/of, for/in.

For loop.

```
my_dict = {  
    'color': 'blue',  
    'fruit': 'apple',  
    'pet': 'dog'  
}  
  
for key in my_dict:  
    print(f"{key} -> {my_dict[key]}")  
  
# color -> blue  
# fruit -> apple  
# pet -> dog
```

Python.

For loop.

```
myObj = {  
    color: 'blue',  
    fruit: 'apple',  
    pet: 'dog'  
}  
  
for (key in myObj) {  
    console.log(` ${key} -> ${myObj[key]}`)  
}  
// color -> blue  
// fruit -> apple  
// pet -> dog
```

JavaScript.

For, for/of, for/in.

Functions.



```
def my_function():
    if True:
        print("It works!")

my_function()
# It works!
```

Python.



```
function myFunction() {
    if (true) {
        console.log("It works!")
    }
}

myFunction()
// It works!
```

JavaScript.

Classes.



```
class Rectangle:  
    sides = 4  
  
    def __init__(self, height, width):  
        self.height = height  
        self.width = width  
  
rect = Rectangle(2, 4)
```



```
class Rectangle {  
    sides = 4;  
  
    constructor(height, width) {  
        this.height = height  
        this.width = width  
    }  
  
rect = new Rectangle(2, 4)
```

Python.

JavaScript.

```
even = [2,4,6]
odd = [1,3,5]

list_1 = [*even, *odd]
print(list_1)
# [2, 4, 6, 1, 3, 5]

list_2 = even + odd
print(list_2)
# [2, 4, 6, 1, 3, 5]
```

Python.

Unpacking Iterables.



```
let even = [2, 4, 6]
let odd = [1, 3, 5]

let array_1 = [...even, ...odd]
console.log(array_1)
// [ 2, 4, 6, 1, 3, 5 ]
```

JavaScript.

Spread Operator.

Unpacking.

```
● ● ●  
test_dict = {'foo': 'bar'}  
  
print({**test_dict, 'foo2': 'bar2'})  
# {'foo': 'bar', 'foo2': 'bar2'}
```

Python.

Unpacking dict.

```
● ● ●  
let test_dict = {foo: 'bar'}  
  
console.log({...test_dict, foo2: 'bar2'})  
// { foo: 'bar', foo2: 'bar2' }
```

JavaScript.

Spread Operator.

F-string.

```
name = "Camila"
n_cats = 3

print(f"Hello! I am {name} and I have {n_cats} cats")
# Hello! I am Camila and I have 3 cats
```

Python.

F-string.



```
let my_name = "Camila"
let n_cats = 3

console.log(`Hello! I am ${my_name} and I have ${n_cats} cats`)
// Hello! I am Camila and I have 3 cats
```

JavaScript.

Template literals.

- 
- 1. Python and JavaScript.**
 - 2. Installation.**
 - 3. REPL.**
 - 4. Syntax.**
 - 5. List comprehension and functional programming.**
 - 6. Pattern matching.**
 - 7. Lambda functions.**
 - 8. Type hints and Typescript.**

List comprehension.

```
languages = [  
    "Java",  
    "Javascript",  
    "Python",  
    "Rust",  
    "Go"  
]  
  
new_list = [f"{language} 🍄"  
            for language in languages  
            if "J" in language]  
  
print(new_list)  
# ['Java 🍄', 'Javascript 🍄']
```

Python.

List comprehension.

```
let languages = [  
    "Java",  
    "Javascript",  
    "Python",  
    "Rust",  
    "Go"  
]  
  
let new_list = languages  
.filter(language => language.includes("J"))  
.map(language => `${language} 🍄`)  
  
console.log(new_list)  
// ['Java 🍄', 'Javascript 🍄']
```

JavaScript.

Map and filter.

List comprehension.

```
languages = [
    "Java",
    "Javascript",
    "Python",
    "Rust",
    "Go"
]

new_list = filter(
    lambda language: "J" in language, languages)
new_list = map(
    lambda language: f"{language} 🍄", new_list)

print(list(new_list))
# ['Java 🍄', 'Javascript 🍄']
```

Python.

Map and filter.

```
let languages = [
    "Java",
    "Javascript",
    "Python",
    "Rust",
    "Go"
]

let new_list = languages
    .filter(language => language.includes("J"))
    .map(language => `${language} 🍄`)

console.log(new_list)
// ['Java 🍄', 'Javascript 🍄']
```

JavaScript.

Map and filter.

- 
- 1. Python and JavaScript.**
 - 2. Installation.**
 - 3. REPL.**
 - 4. Syntax.**
 - 5. List comprehension and functional programming.**
 - 6. Pattern matching.**
 - 7. Lambda functions.**
 - 8. Type hints and Typescript.**

Pattern matching.

```
def get_url_conf(host):
    mapping = {
        "www.example-a.dev": "firstApp.urls",
        "www.example-b.dev": "secondApp.urls",
        "www.example-c.dev": "thirdApp.urls"
    }

    return mapping.get(host, "Sorry, no match")

print(get_url_conf("www.example-a.dev"))
# firstApp.urls

print(get_url_conf("www.example.dev"))
# Sorry, no match
```

```
function getUrlConf(host) {
    switch (host) {
        case "www.example-a.dev":
            return "firstApp.urls"
        case "www-example-b.dev":
            return "secondApp.urls"
        case "www.example-c.dev":
            return "thirdApp.urls"
        default:
            return "Sorry, no match"
    }
}

console.log(getUrlConf("www.example-a.dev"))
// firstApp.urls

console.log(getUrlConf("www.example.dev"))
// Sorry, no match
```

Python.

No switch, dictionary pattern.

JavaScript.

Template literals.



```
match subject:  
    case <pattern_1>:  
        <action_1>  
    case <pattern_2>:  
        <action_2>  
    case <pattern_3>:  
        <action_3>  
    case _:  
        <action_wildcard>
```

Pattern matching.

Pattern matching (>=3.10).

Structural pattern matching has been added in the form of a match statement and case statements of patterns with associated actions.

<https://docs.python.org/3.10/whatsnew/3.10.html#pep-634-structural-pattern-matching>.

- 
- 1. Python and JavaScript.**
 - 2. Installation.**
 - 3. REPL.**
 - 4. Syntax.**
 - 5. List comprehension and functional programming.**
 - 6. Pattern matching.**
 - 7. Lambda functions.**
 - 8. Type hints and Typescript.**

Lambda functions.

```
def factory(n):
    return lambda a : a * n

double = factory(2)
triple = factory(3)

print(double(10))
# 20

print(triple(10))
# 30
```

```
function factory (n) {
    return (a) => a * n
}

const double = factory(2)
const triple = factory(3)

console.log(double(10))
// 20

console.log(triple(10))
// 30
```

Python.

Lambda functions.

JavaScript.

Arrow functions.

- 
- 1. Python and JavaScript.**
 - 2. Installation.**
 - 3. REPL.**
 - 4. Syntax.**
 - 5. List comprehension and functional programming.**
 - 6. Pattern matching.**
 - 7. Lambda functions.**
 - 8. Type hints and Typescript.**



```
def greetings(name: str) -> str:  
    return f"Hello {name}"
```

TypeScript becomes JavaScript via the delete key.

```
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

TypeScript file.

```
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

Types are removed.

```
function verify(result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

JavaScript file.

https://github.com/camigomezdev/
python_for_javascripters

done.

JavaScript for Pythoners (and vice versa)