Hello.

JavaScript for Pythoners (and vice versa)

Nov 29, 2023.

About me.

Cami Gómez.

@camigomezdev

About me.

Agenda.

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

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.

Agenda.

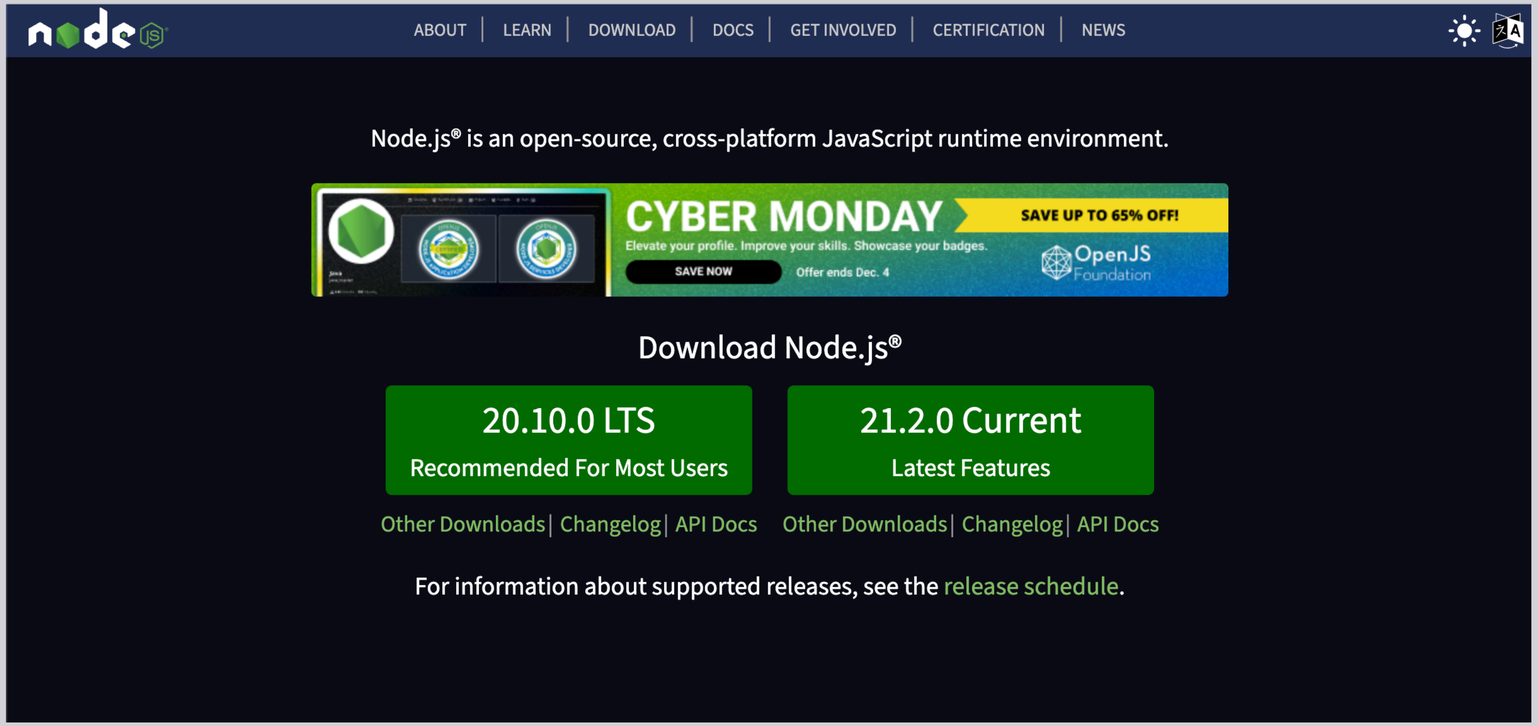
2

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.

Python.

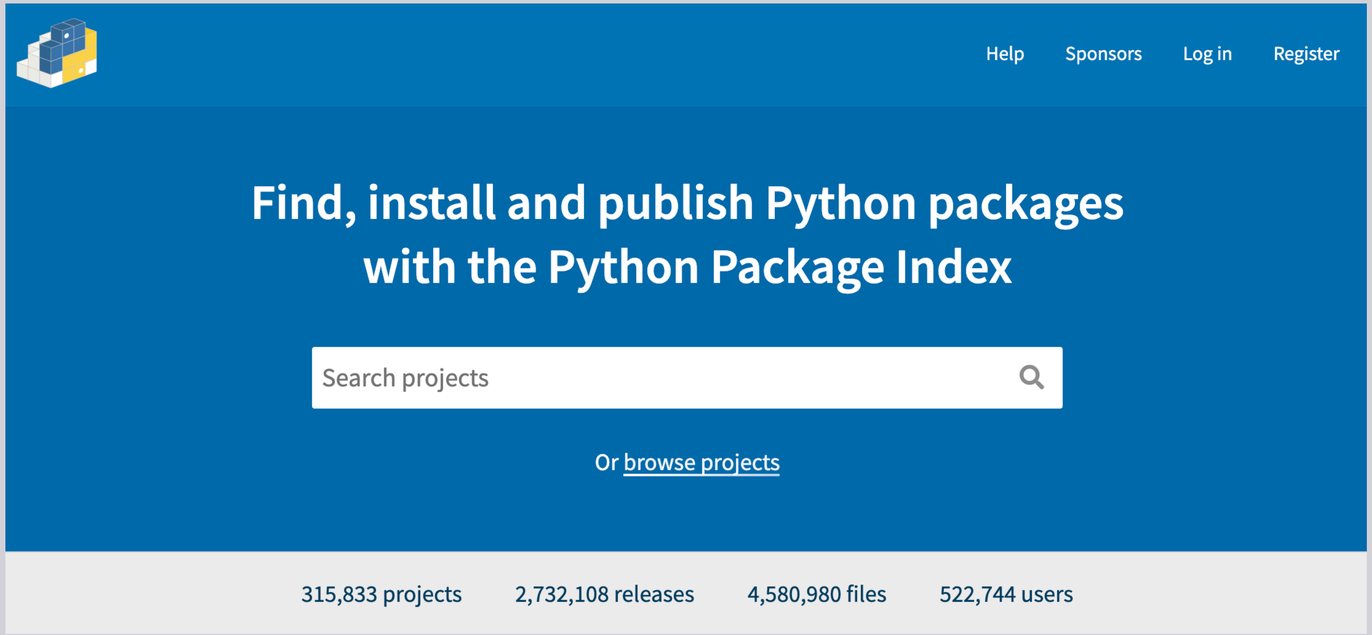
https://www.python.org/downloads/

JavaScript.



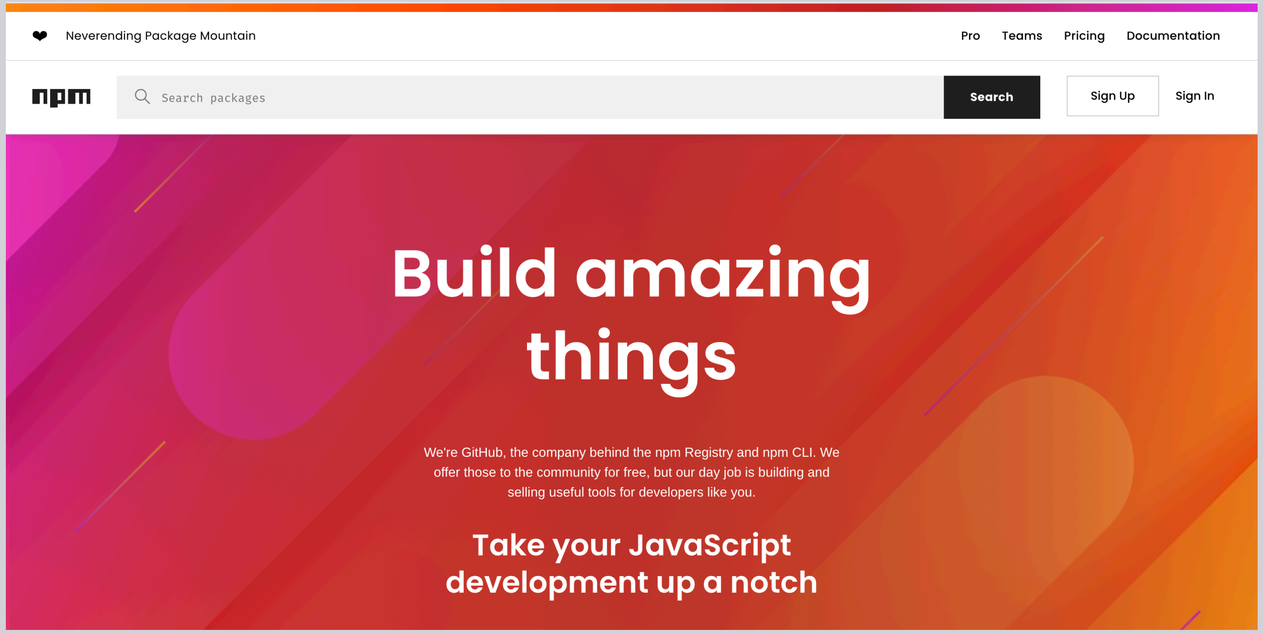
https://nodejs.org/en/

PyPI and Pip.



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

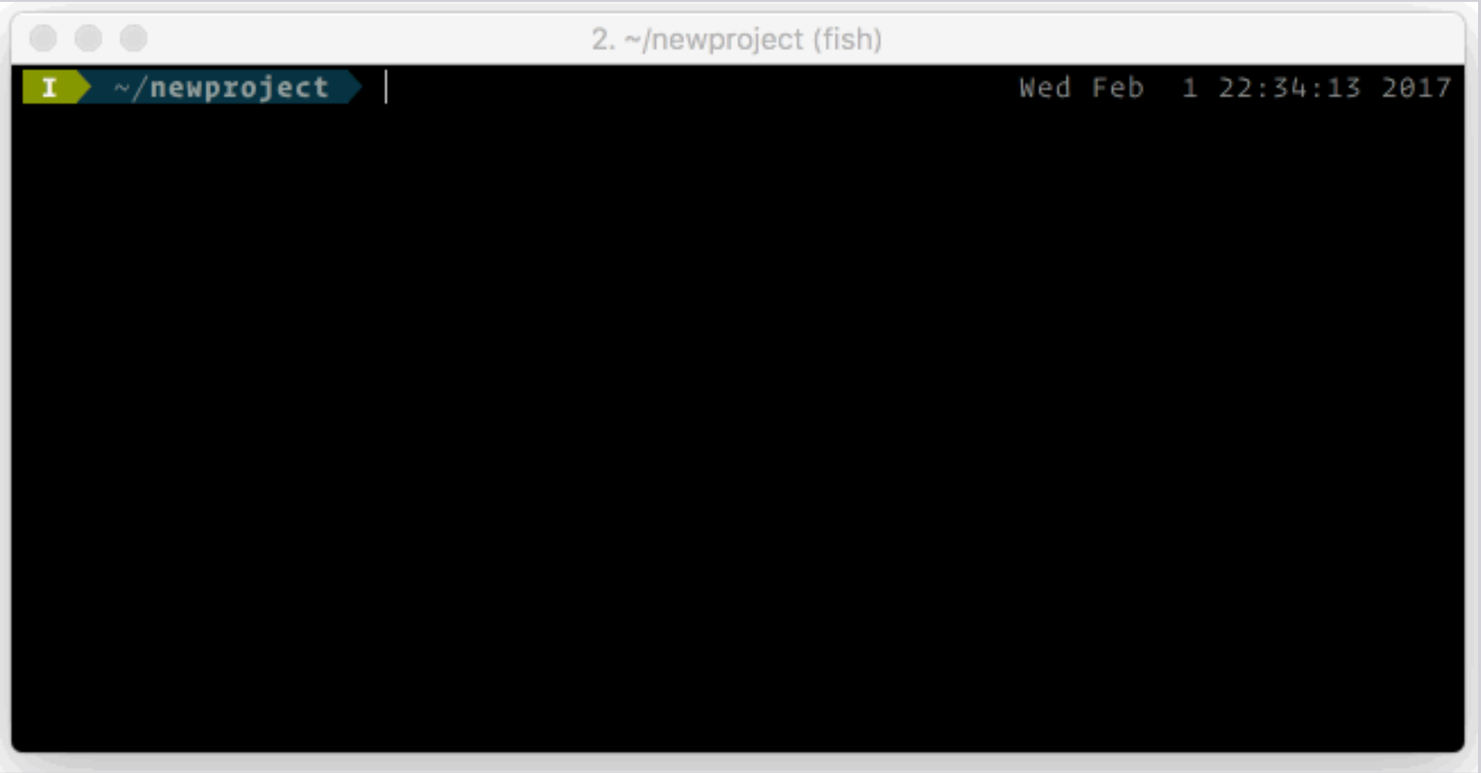
NPM.



npm is a package manager for the JavaScript programming language.

npm is the default package manager for the JavaScript runtime

Pipenv: Python Dev Workflow for Humans.



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

NVM.

Intro

`nvm` allows you to quickly install and use different versions of node via the command line.

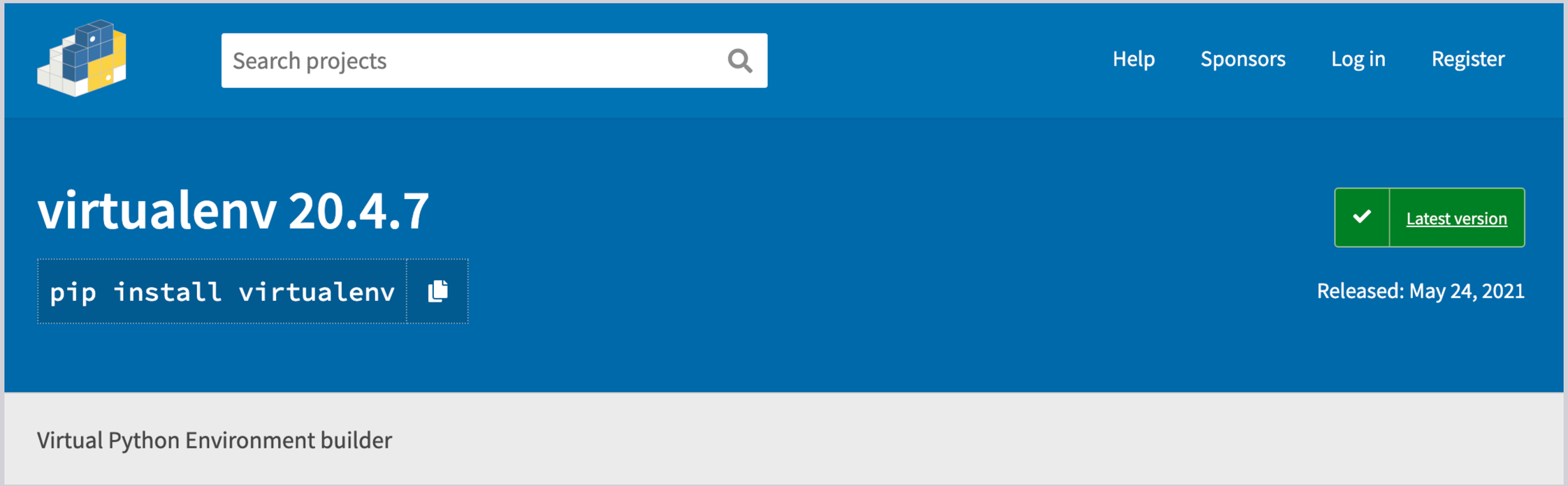
Example:

```
$ 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
```

Simple as that!

Node Version Manager.

Virtualenv.



Creates isolated Python environments.

Python.	JavaScript.
PyPI.	npm (site).
Pip.	npm (command).
Virtualenv.	nvm.
Pipenv.	Node project with nvm.

3

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.

REPL.

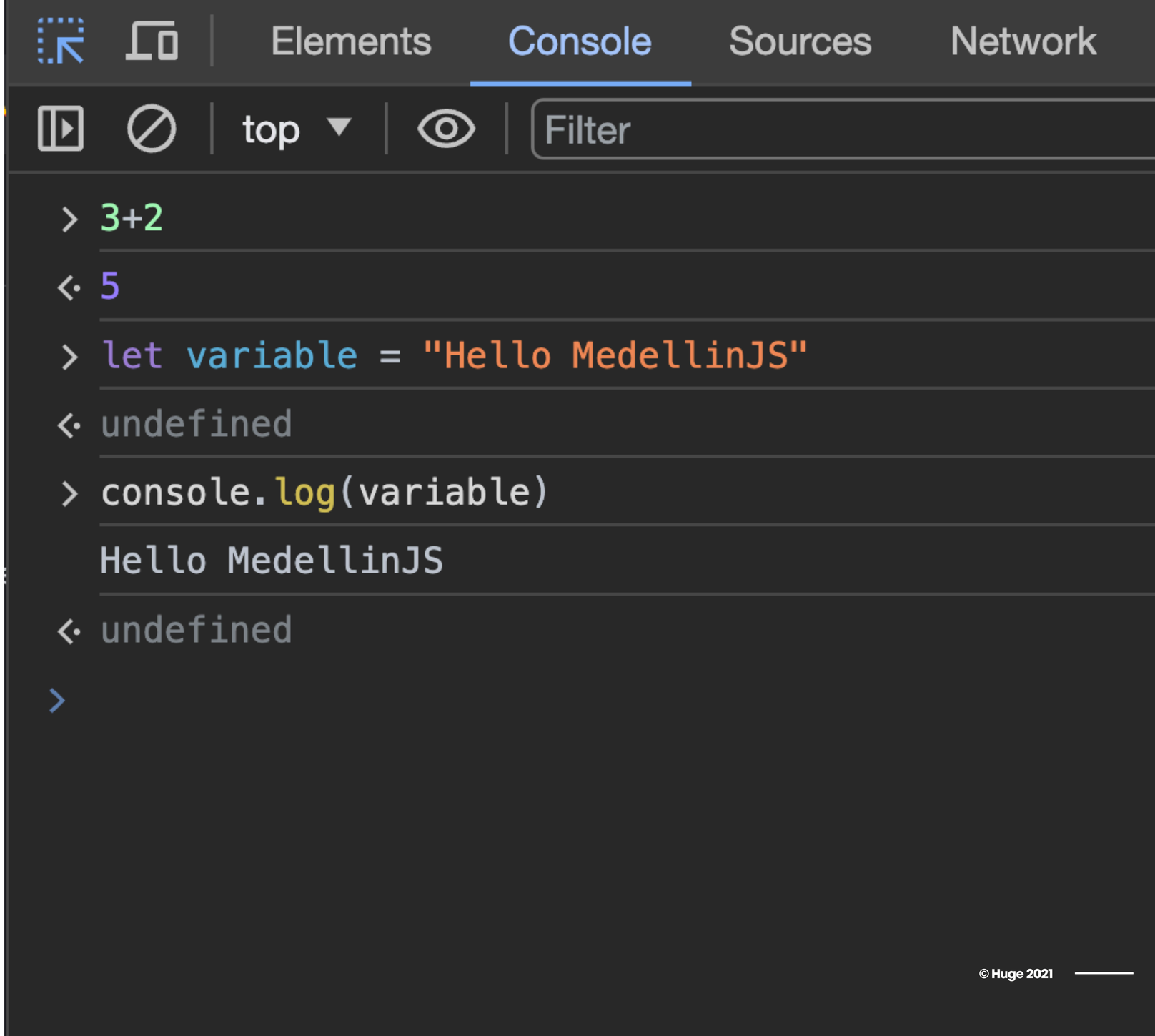
Read, Evaluate, Print, Loop.

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.

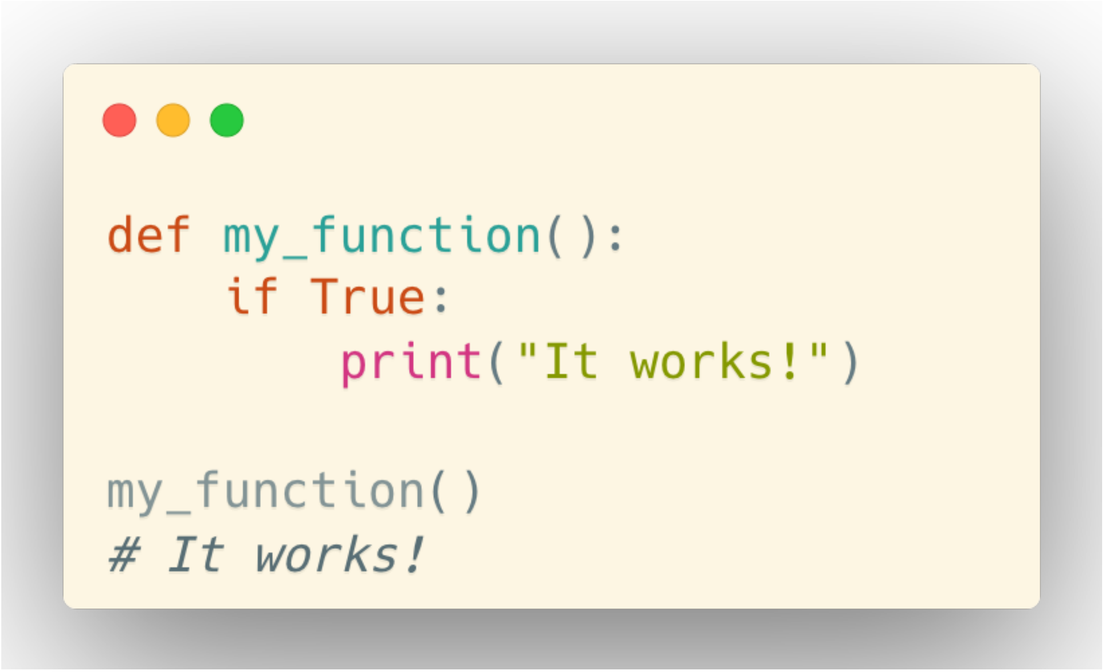


Node.js REPL.

```
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
>
```

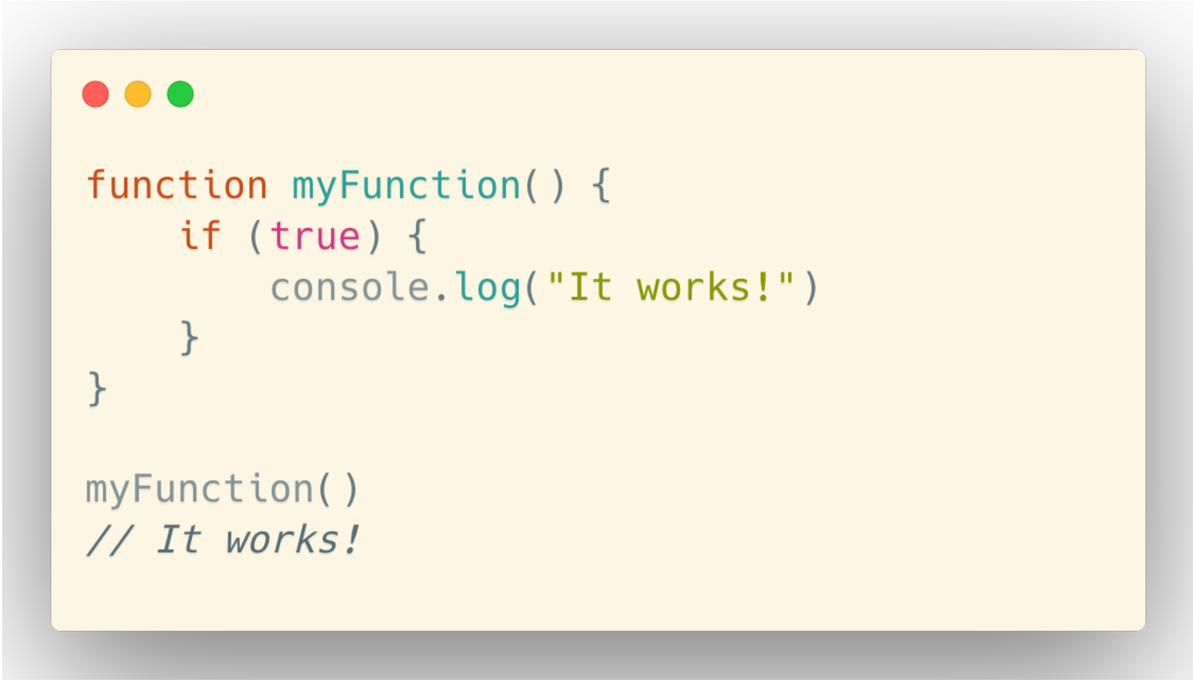
Agenda.

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.



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

my_function()
# It works!
```

Python.



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

myFunction()
// It works!
```

JavaScript.

Arithmetic operators.

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

Comparison operators.

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	!==

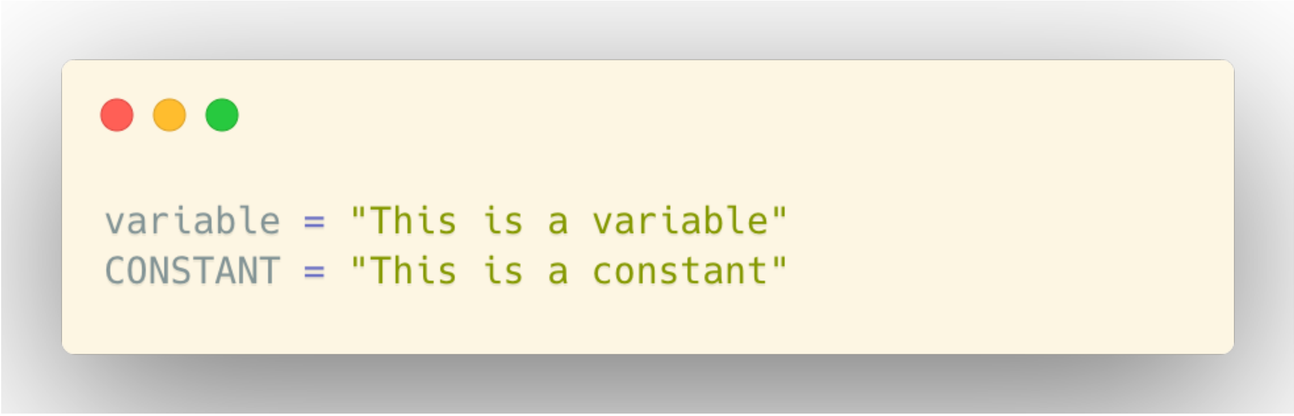
Logical operators.

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

Data types.

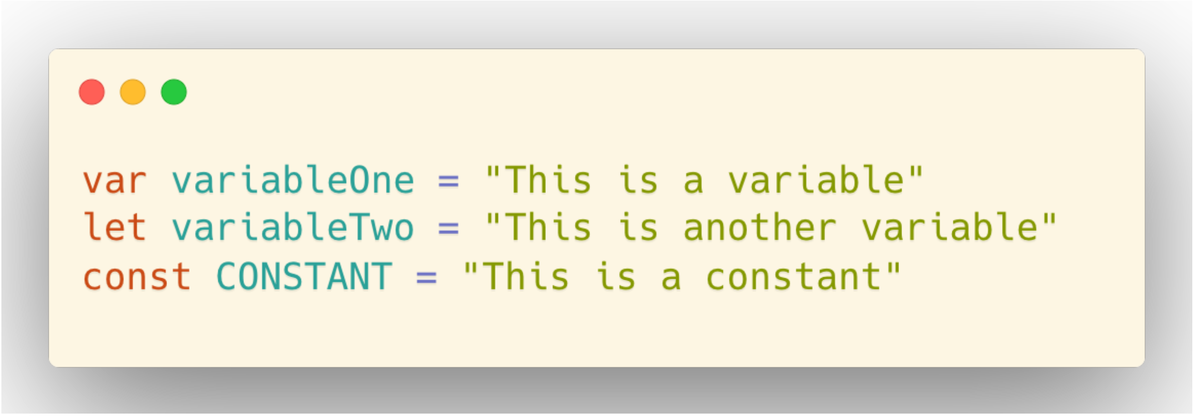
Python.	JavaScript.
float.	Number.
int.	BigInt.
int.	Number.
string.	String.
boolean.	Boolean.
None.	Null.
N/a.	Undefined.

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



```
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"
```

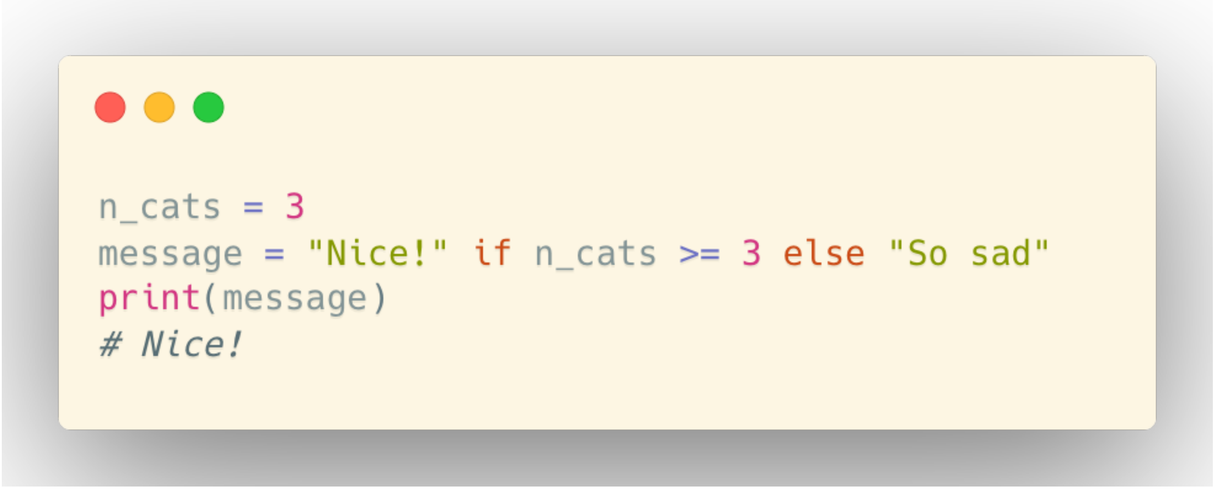
JavaScript.

```python
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
```

Python.

```javascript
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!
```

JavaScript.

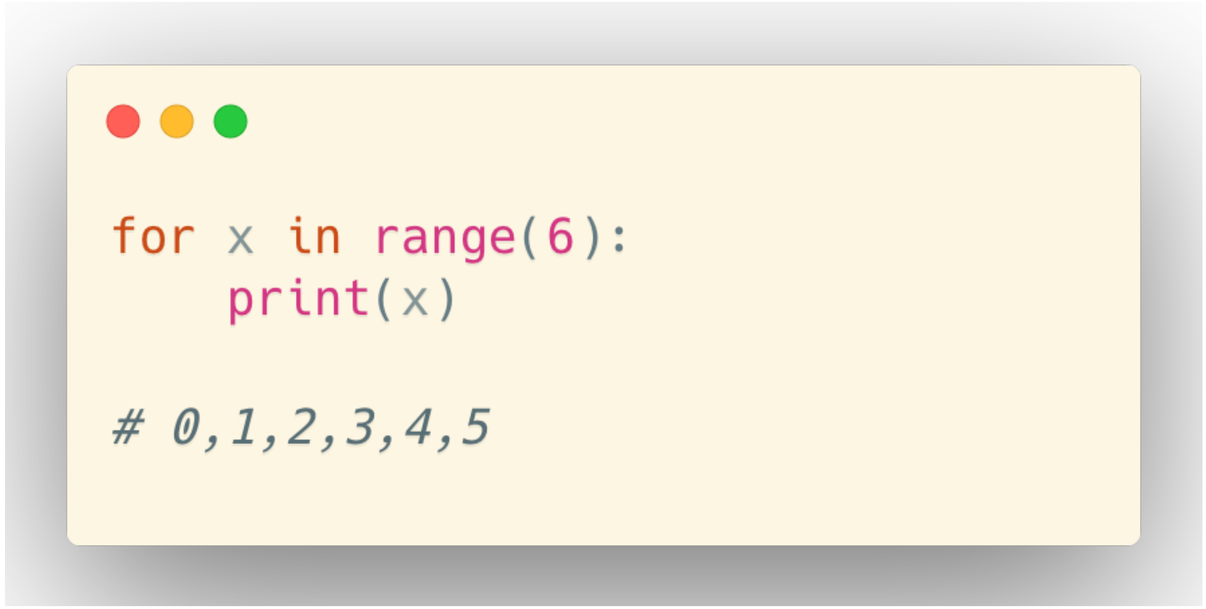


```python
n_cats = 3
message = "Nice!" if n_cats >= 3 else "So sad"
print(message)
# Nice!
```

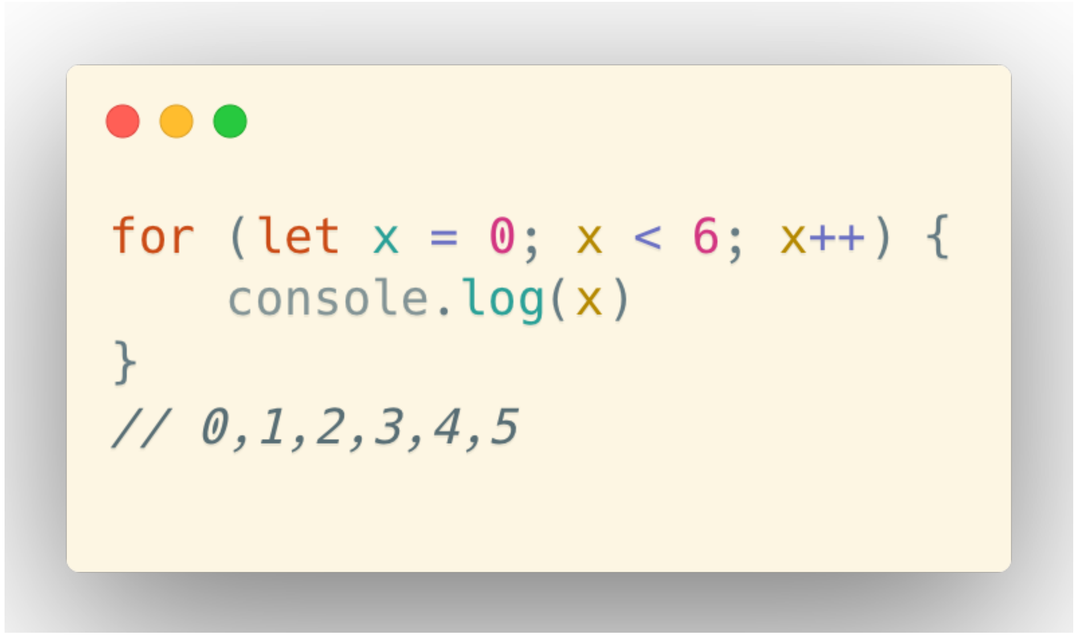
```javascript
let n_cats = 3
let message = n_cats >= 3 ? "Nice!" : "So sad"
console.log(message)
// Nice!
```

Python.

JavaScript.



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

# 0,1,2,3,4,5
```



```javascript
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.

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

# 1,2,3,4
```

Python.

For loop.

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

JavaScript.

For, for/of, for/in.

```python
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
```

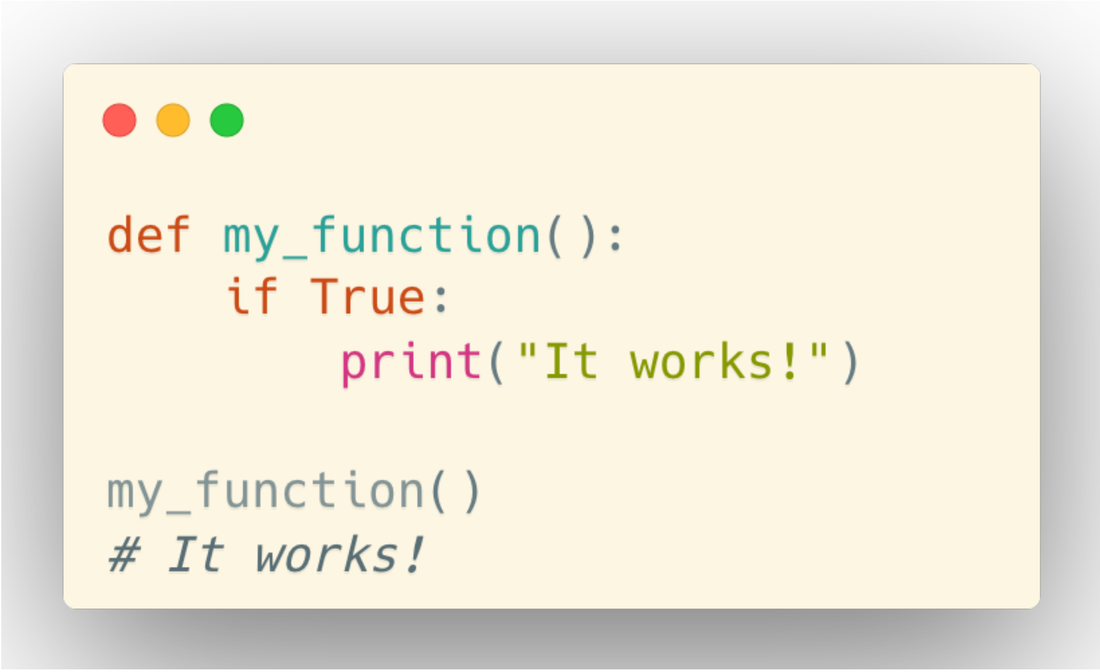
Python.

For loop.

```javascript
myObj = {
    color: 'blue',
    fruit: 'apple',
    pet: 'dog'
}

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

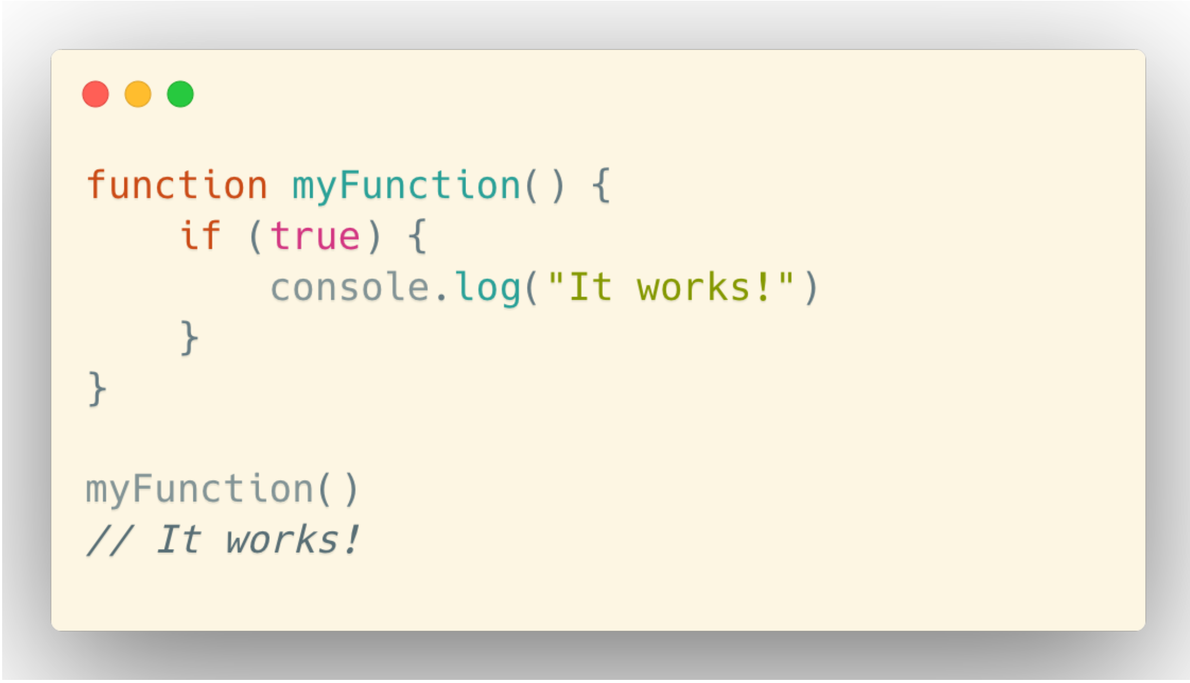
JavaScript.

For, for/of, for/in.



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

my_function()
# It works!
```

Python.



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

myFunction()
// It works!
```

JavaScript.

```python
class Rectangle:
    sides = 4

    def __init__(self, height, width):
        self.height = height
        self.width = width


rect = Rectangle(2, 4)
```

Python.

```javascript
class Rectangle {
    sides = 4;

    constructor(height, width) {
        this.height = height
        this.width = width
    }
}

rect = new Rectangle(2, 4)
```

JavaScript.

```python
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.

```javascript
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.

```python
test_dict = {'foo': 'bar'}

print({**test_dict, 'foo2': 'bar2'})

# {'foo': 'bar', 'foo2': 'bar2'}
```

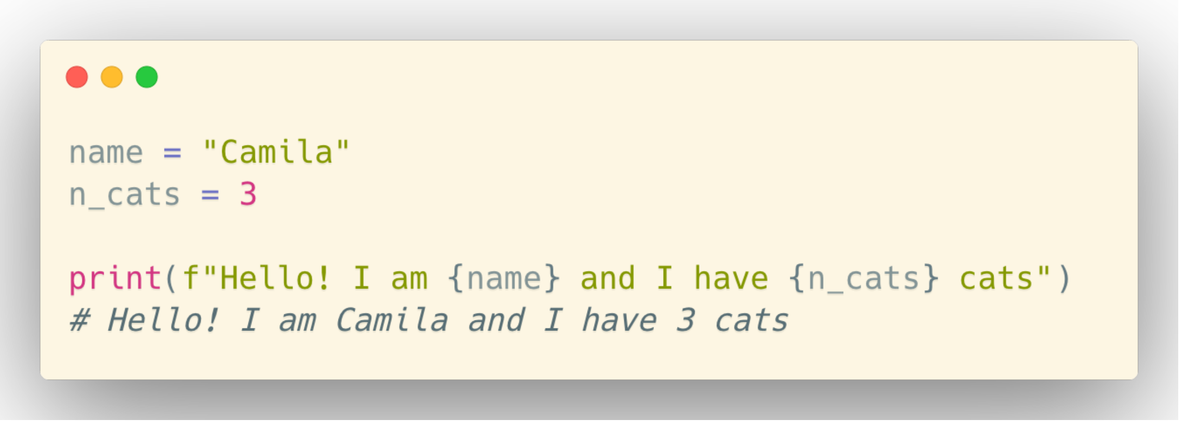
Python.

Unpacking dict.

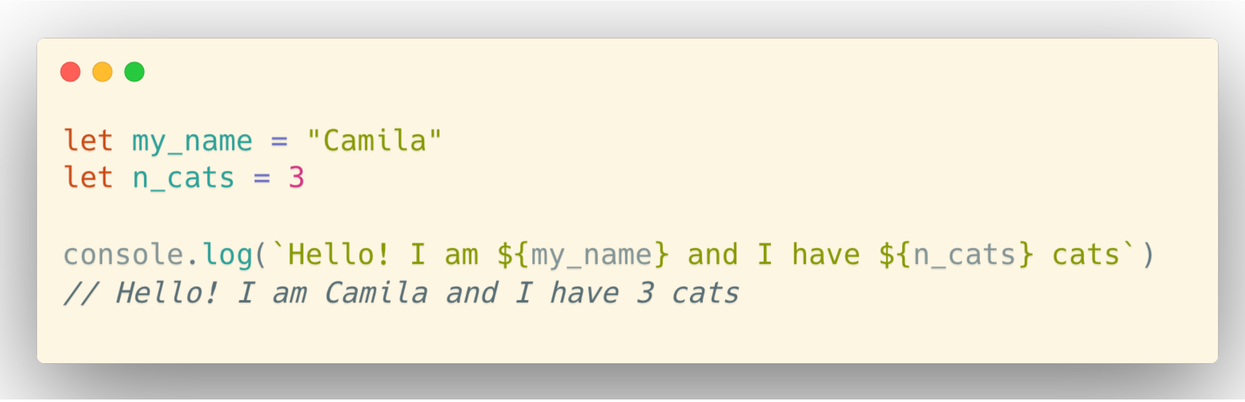
```javascript
let test_dict = {foo: 'bar'}

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

JavaScript.

Spread Operator.



```python
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
```



```javascript
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
```

Python.

F-string.

JavaScript.

Template literals.

Agenda.

5

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.

```python
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.

```javascript
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.

```python
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.

```javascript
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.

Agenda.

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.

```python
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
```

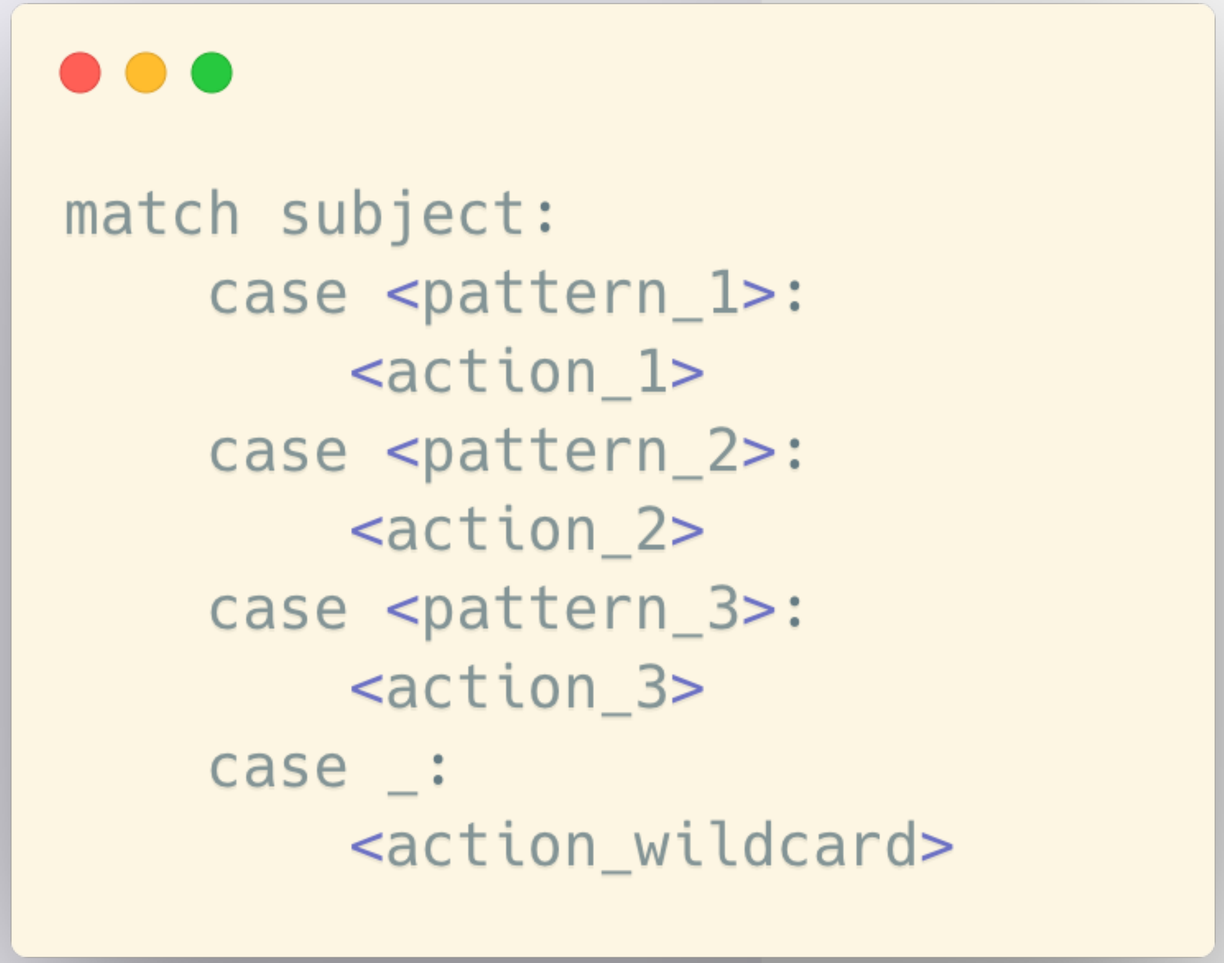
Python.

No switch, dictionary pattern.

```javascript
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
```

JavaScript.

Template literals.



Pattern matching (>=3.10).

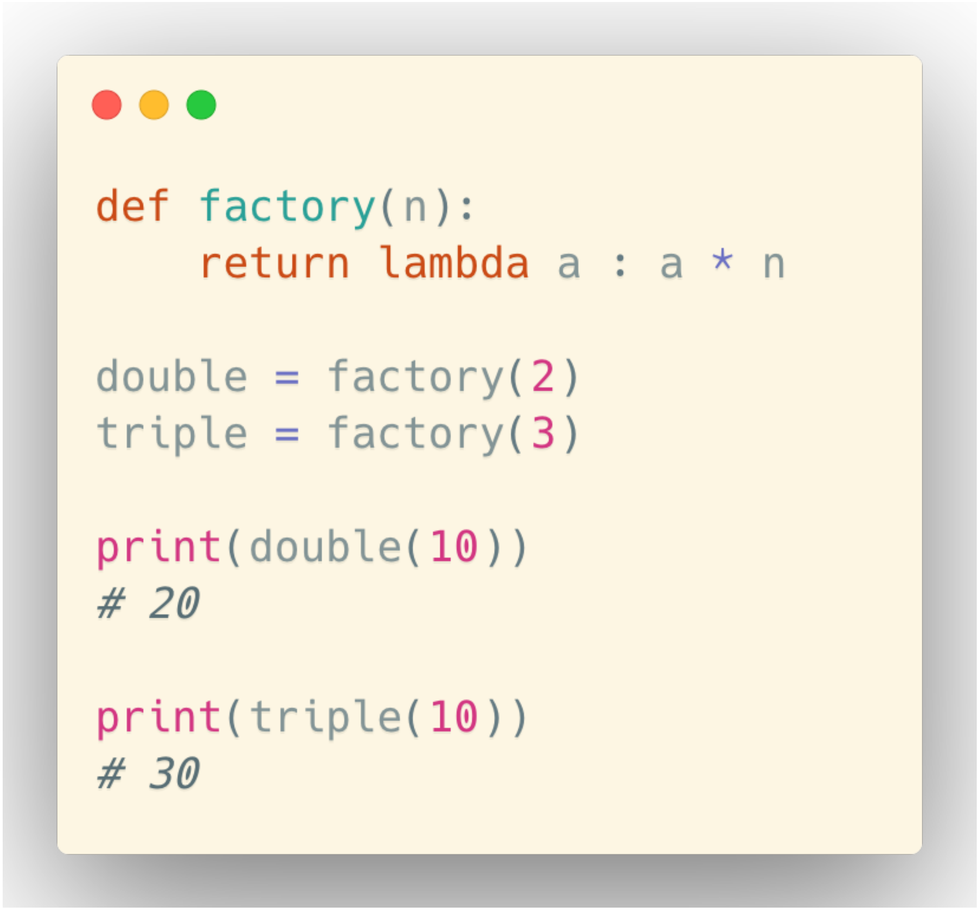
Structural pattern matching has been added in the form of a match statement and case statements of patterns with associated actions.

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

Agenda.

7

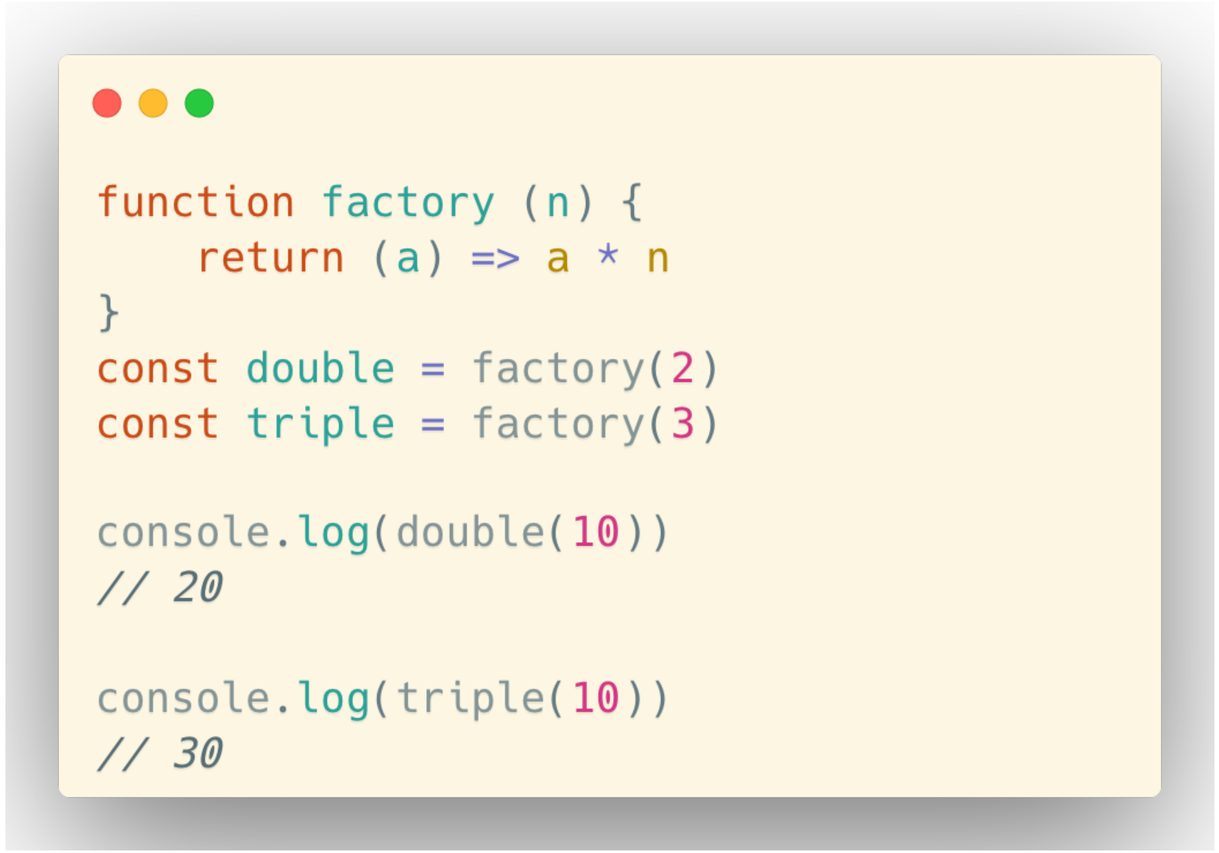
1. Python and JavaScript.
2. Installation.
3. REPL.
4. Syntax.
5. List comprehension and functional programming.
6. Pattern matching.
7. Lambda functions.
8. Type hints and Typescript.



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

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

print(double(10))
# 20

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

Python.

Lambda functions.



```javascript
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
```

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.

Type hints.

TypeScript becomes JavaScript via the delete key.

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

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

TypeScript file.

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

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

Types are removed.

```ts
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)

Nov 29, 2023.