Moduler, rekursion och generatorer

Daniel Bosk

KTH EECS

18th September 2020

import module

Example (bad-module.py)

```
1 """This is a bad module"""
2
3 def hello_world():
4     """Prints 'Hello, world!'"""
5     print("Hello, world!")
6
7 hello_world()
```

Example (test-good-bad.py)

```
1 """Tests good and bad modules"""
2
3 import bad_module
4
5 bad_module.hello_world()
```

Example (good-module.py)

```
1  """This is a good module"""
2
3  def hello_world():
4      """Prints 'Hello, world!!!!!!'"""
5      print("Hello, world!!!!!!!!!!")
6
7  def main():
8      hello_world()
9
10 if __name__ == "__main__":
11     main()
```

Example (input-type.py, del 1)

```
1  """Take input more easily."""
2
3  def input_type(t, prompt=""):
4      """Take input, convert to type t; repeat if error."""
5      while True:
6          try:
7              return t(input(prompt))
8          except ValueError:
9              if t == int:
10                 print(f"Sorry, can't convert to integer.")
11             else:
12                 print(f"Sorry, can't convert to {t}.")
```

Example (input-type.py, del 2)

```
14  def main():
15      """Test functionality of this module"""
16      x = input_type(int, "x = ")
17      y = input_type(int, "y = ")
18      z = input_type(float, "z = ")
19      name = input_type(str, "Your name: ")
20
21      print(f"{x} + {y} = {x+y}")
22      print(f"z = {z}")
23      print(f"Your name is {name}")
24
25  if __name__ == "__main__":
26      main()
```

Example (Använda modulen)

```
1 import input_type
2
3 x = input_type(int, "x = ")
4 print(f"x = {x}")
```

```
$ pylint prog.py
```

```
def f(x):
    if p(x):
        return c
    else:
        return f(g(x))
```

Example (factorial.py, del 1)

```
1  """Calculate factorial"""
2
3  import input_type
4
5  def factorial(num):
6      """returns the factorial of num"""
7      if num == 1:
8          return 1
9      elif num < 1:
10         raise ArithmeticError("Must be positive numbers")
11     return num*factorial(num-1)
```

Example (factorial.py, del 2)

```
12
13  def main():
14      """test the functions in this module"""
15      num = input_type(int, "Enter a number: ")
16      print(f"{num}! = {factorial(num)}")
17
18  if __name__ == "__main__":
19      main()
```

Example (search.py, del 1)

```
5  def in_list(item, lst):
6      """ Check if item is in lst """
7      if len(lst) < 1:
8          return False
9      elif len(lst) == 1:
10         return lst[0] == item
11
12     middle = len(lst)//2
13     if item == lst[middle]:
14         return True
15     elif item < lst[middle]:
16         half_lst = lst[:middle] # lst[0:middle]
17     else:
18         half_lst = lst[middle+1:]
19     return in_list(item, half_lst)
```

Example (search.py, del 2)

```
 3  import input_type

    ...

21
22  def main():
23      """Test functionality"""
24      l = sorted([1, 2, 4, 5, 8, 10, 18, 20, 21, 22, 30])
25      x = input_type.input_type(int, "What to search for? ")
26      if in_list(x, l):
27          print("Yes!")
28      else:
29          print("No!")
30
31  if __name__ == "__main__":
32      main()
```

```
def gen():
    i = 0
    while True:
        yield i
        i += 1
```

Example (filter-own.py)

```
 7  def own_filter(f, lst):
 8      result = []
 9      for i in lst:
10          if f(i):
11              result.append(i)
12      return result
```

Example (filter-gen.py)

```
 7  def own_filter(f, lst):
 8      """filter elements of lst based on f"""
 9      for i in lst:
10          if f(i):
11              yield i
```

Example (filter-own.py)

```
7  def own_filter(f, lst):
8      result = []
9      for i in lst:
10         if f(i):
11             result.append(i)
12     return result
```

Example (filter-gen.py)

```
7  def own_filter(f, lst):
8      """filter elements of lst based on f"""
9      for i in lst:
10         if f(i):
11             yield i
```

Example (mapping-own.py)

```
16 def own_map(f, lst):
17     result = []
18     for i in lst:
19         result.append(f(i))
20     return result
```

Example (mapping-gen.py)

```
16 def own_map(f, lst):
17     """map i to f(i)"""
18     for i in lst:
19         yield f(i)
```

Example (mapping-own.py)

```
16 def own_map(f, lst):
17     result = []
18     for i in lst:
19         result.append(f(i))
20     return result
```

Example (mapping-gen.py)

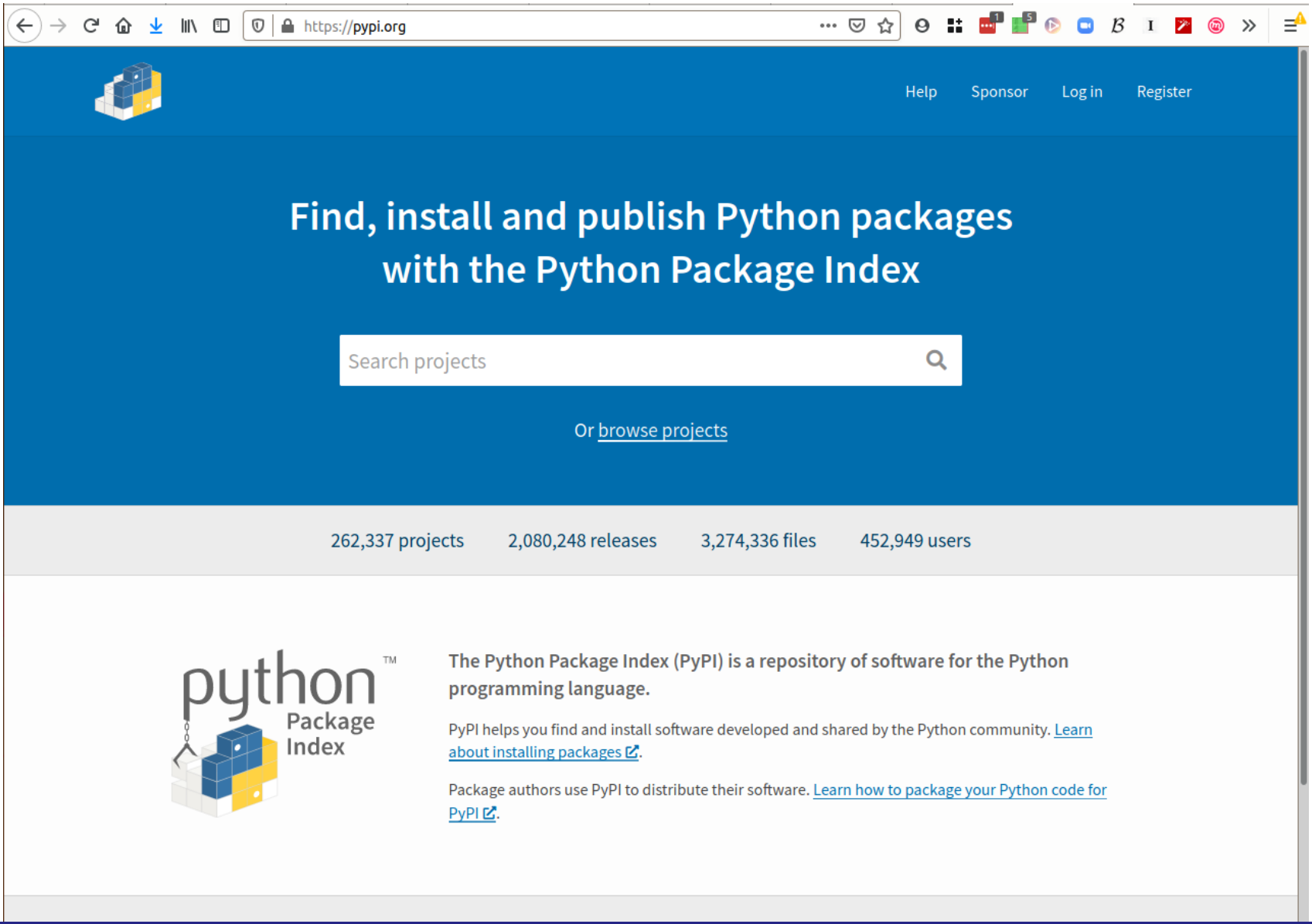
```
16 def own_map(f, lst):
17     """map i to f(i)"""
18     for i in lst:
19         yield f(i)
```

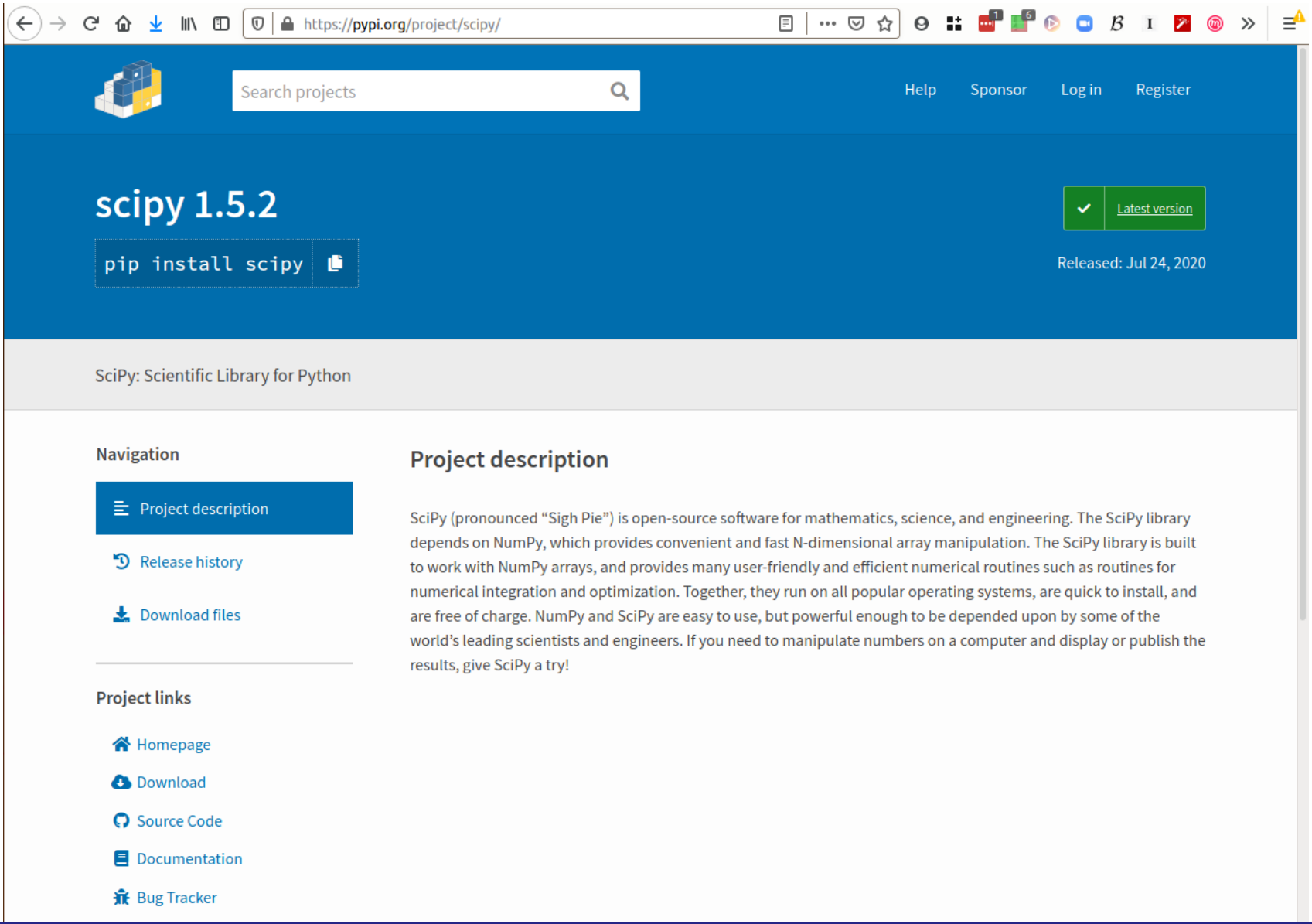
Example (gen-adv.py, del 1)

```
3  def map_nogen(func, lst):
4      result = []
5      for i in lst:
6          print(f"nogen {i} -> {func(i)}")
7          result.append(func(i))
8      return result
9
10 def map_gen(func, lst):
11     for i in lst:
12         print(f"gen {i} -> {func(i)}")
13         yield func(i)
```

Example (gen-adv.py, del 2)

```
15  def main():
16      l = [1, 2, 3, 4, 5]
17      f = lambda x: 2*x
18      for i in map_nogen(f, l):
19          print(i)
20          break
21      for i in map_gen(f, l):
22          print(i)
23          break
24
25  if __name__ == "__main__":
26      main()
```

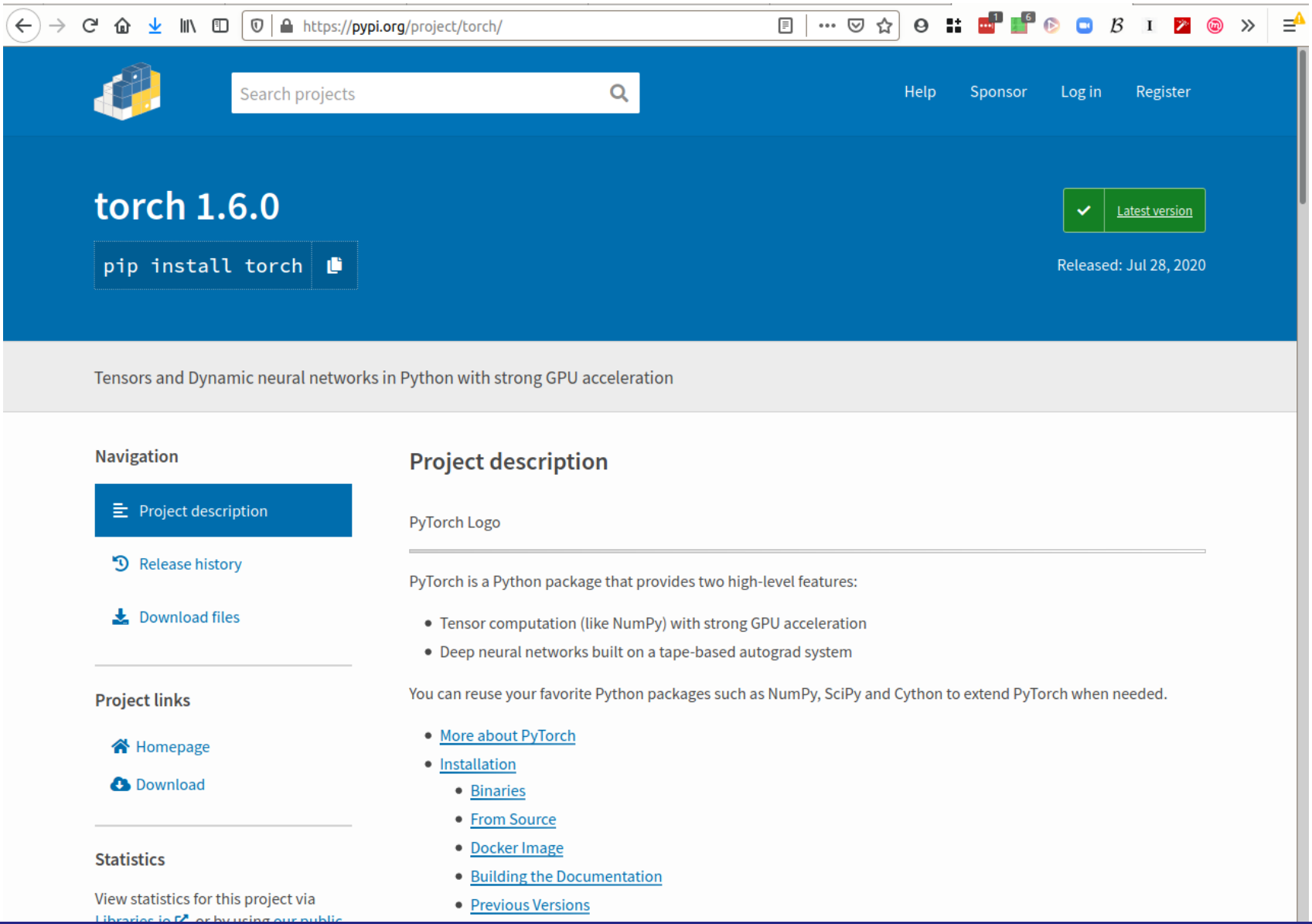

https://pypi.org
Help
Sponsor
Log in
Register
Find, install and publish Python packages with the Python Package Index
Search projects
Or browse projects
262,337 projects
2,080,248 releases
3,274,336 files
452,949 users
python Package Index
The Python Package Index (PyPI) is a repository of software for the Python programming language.
PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.
Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.


https://pypi.org/project/scipy/
Search projects
Help
Sponsor
Log in
Register
scipy 1.5.2
pip install scipy
Latest version
Released: Jul 24, 2020
SciPy: Scientific Library for Python
Navigation
Project description
Release history
Download files
Project links
Homepage
Download
Source Code
Documentation
Bug Tracker
Project description
SciPy (pronounced “Sigh Pie”) is open-source software for mathematics, science, and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install, and are free of charge. NumPy and SciPy are easy to use, but powerful enough to be depended upon by some of the world’s leading scientists and engineers. If you need to manipulate numbers on a computer and display or publish the results, give SciPy a try!


https://pypi.org/project/matplotlib/
Search projects
Help
Sponsor
Log in
Register
matplotlib 3.3.2
pip install matplotlib
Latest version
Released: Sep 15, 2020
Python plotting package
Navigation
Project description
Release history
Download files
Project links
Homepage
Download
Donate
Forum
Bug Tracker
Project description
pypi package 3.3.2
downloads/month 9M
powered by NumFOCUS
help forum discourse
chat on gitter
issue tracking github
PR Welcome
build passing
Azure Pipelines succeeded
build failing
codecov 83%
code quality: python A
matplotlib
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
Check out our home page for more information.

Example (Installation)

```
$ pip install numpy scipy matplotlib
```


https://pypi.org/project/torch/
Search projects
Help
Sponsor
Log in
Register
torch 1.6.0
pip install torch
Latest version
Released: Jul 28, 2020
Tensors and Dynamic neural networks in Python with strong GPU acceleration
Navigation
Project description
Release history
Download files
Project links
Homepage
Download
Statistics
View statistics for this project via
Project description
PyTorch Logo
PyTorch is a Python package that provides two high-level features:
Tensor computation (like NumPy) with strong GPU acceleration
Deep neural networks built on a tape-based autograd system
You can reuse your favorite Python packages such as NumPy, SciPy and Cython to extend PyTorch when needed.
More about PyTorch
Installation
Binaries
From Source
Docker Image
Building the Documentation
Previous Versions