

# TUTORIAL ON VSCODE-PVS

PAOLO MASCI  
PAOLO.MASCI@NIA.NET.ORG

NATIONAL INSTITUTE OF AEROSPACE  
LANGLEY RESEARCH CENTER

MAY 2022

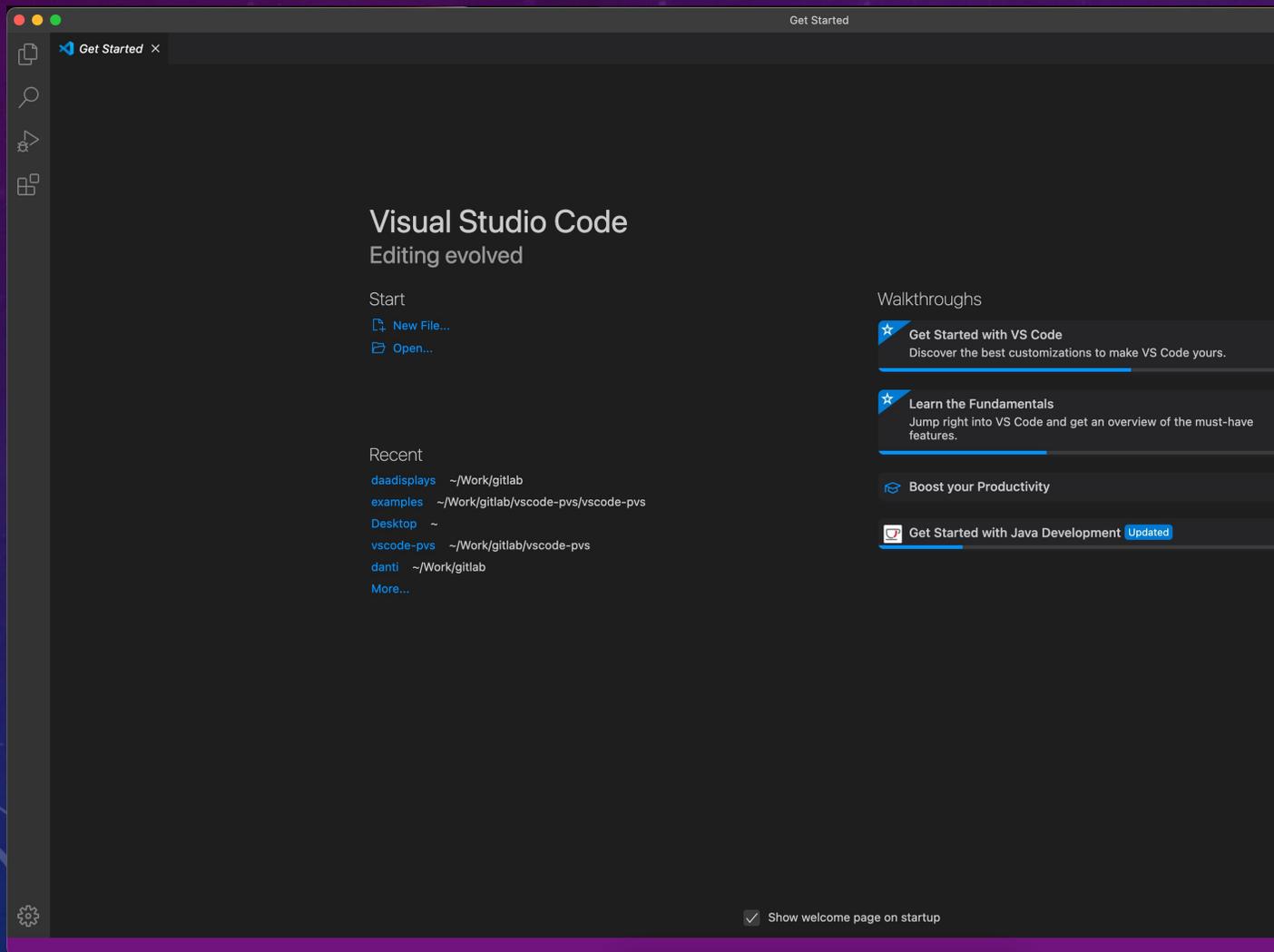
# TOPICS COVERED IN THIS TUTORIAL

1. Installation of VSCode-PVS
2. Creation and editing of PVS theories
3. Typechecking and debugging PVS theories
4. Development of PVS proofs in VSCode-PVS
5. Documenting your PVS files
6. Prototyping functions provided by VSCode-PVS

# INSTALLATION OF VSCODE-PVS

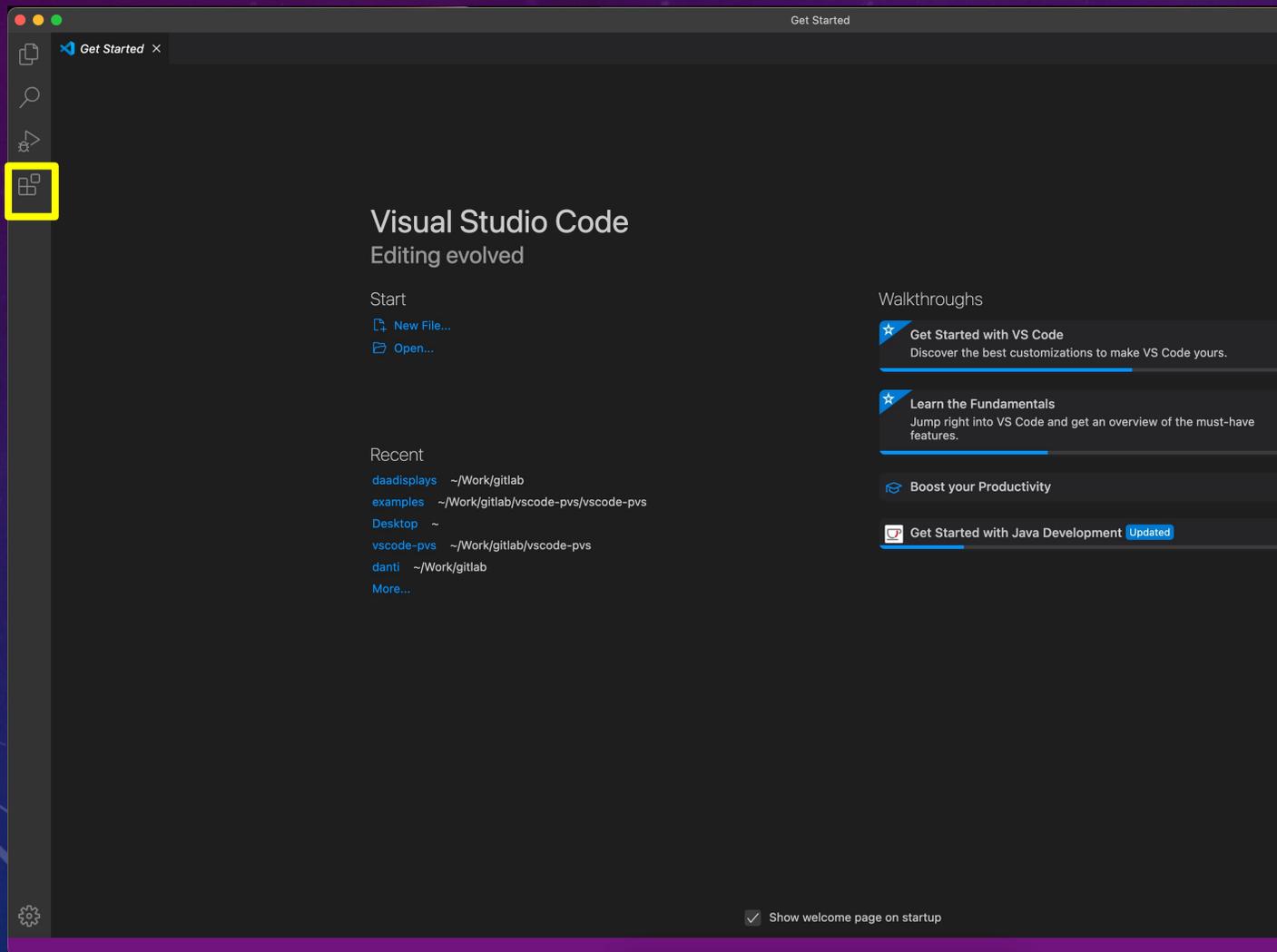
- VSCode-PVS can be installed from the Visual Studio Code marketplace
- Requirements
  - Visual Studio Code (<https://code.visualstudio.com/download>)
  - NodeJS (<https://nodejs.org/en/download/>)
  - Linux or Intel Mac

# INSTALLATION OF VSCODE-PVS



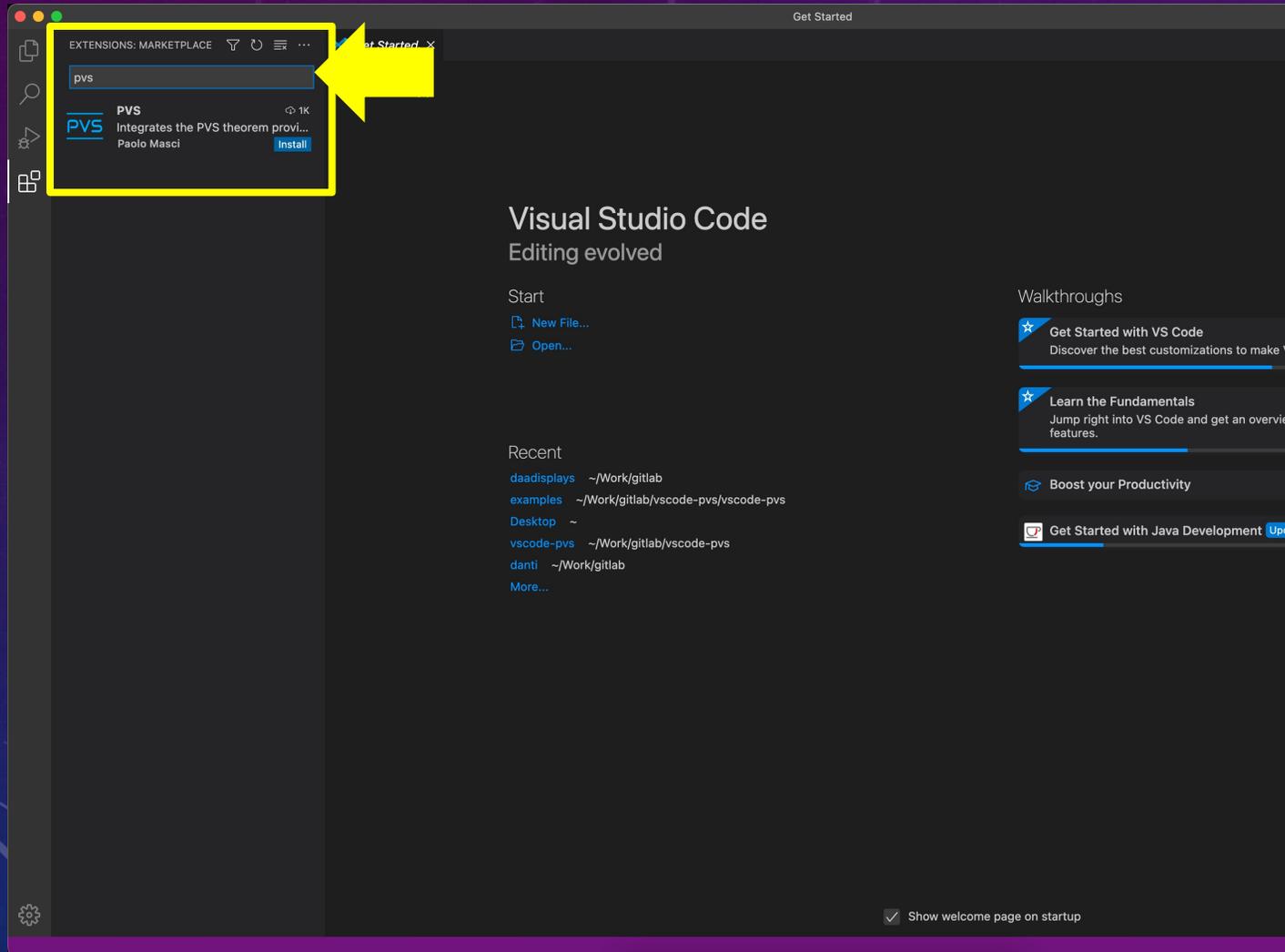
## 1. Open Visual Studio Code

# INSTALLATION OF VSCODE-PVS



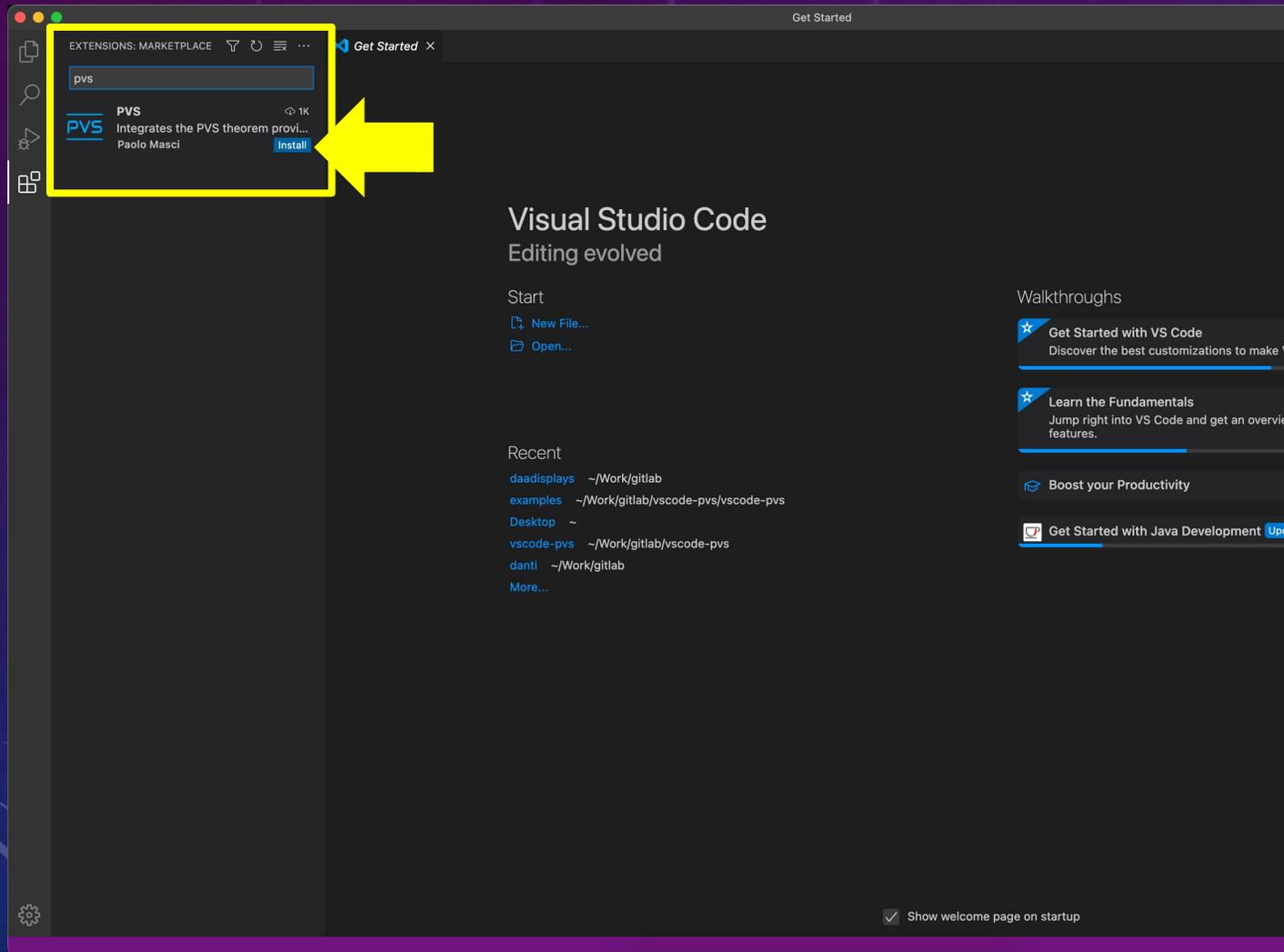
1. Open Visual Studio Code
2. Click on the Extensions icon

# INSTALLATION OF VSCODE-PVS



1. Open Visual Studio Code
2. Click on the Extensions icon
3. Search PVS in the Marketplace

# INSTALLATION OF VSCODE-PVS



1. Open Visual Studio Code
2. Click on the Extensions icon
3. Search PVS in the Marketplace
4. Click Install

# INSTALLATION OF VSCODE-PVS

Welcome Screen

PVS Extension Icon

The screenshot displays the Visual Studio Code interface with the Extensions Marketplace open. The 'PVS' extension by Paolo Masci is highlighted. A yellow box around the extension icon in the sidebar is labeled 'PVS Extension Icon'. A yellow arrow points from this icon to the 'Welcome Screen' label. Another yellow arrow points from the 'Welcome Screen' label to the extension details page. The details page for 'PVS v1.0.61' shows it has 1,548 installs and a 5-star rating. It includes buttons for 'Set File Icon Theme', 'Disable', and 'Uninstall'. Below this, there are tabs for 'Details', 'Feature Contributions', and 'Runtime Status'. The main content area features the title 'VSCode-PVS: An Integrated Development Environment for the Prototype Verification System' and a description: 'VSCode-PVS is a new integrated development environment for creating, evaluating and verifying PVS specifications. The environment redefines the way developers interact with PVS, and better aligns the PVS front-end to the functionalities provided by development environments used for programming languages such as C++ and Java.' At the bottom, there is a preview of the PVS workspace explorer and proof explorer, showing a list of files and a detailed view of a proof step.

# PVS ALLEGRO + NASALIB

VSCoDe-PVS will check if PVS Allegro and NASALib are already present on your system (if they are not present, they will be download by VSCoDe-PVS)

Default directory structure when you choose your home folder as base folder for the installation of PVS Allegro

- PVS Allegro: ~/pvs-7.1.0
- NASALib: ~/pvs-7.1.0/nasalib
- Your PVS developments: ~/Workspaces

# CREATION AND EDITING OF PVS THEORIES

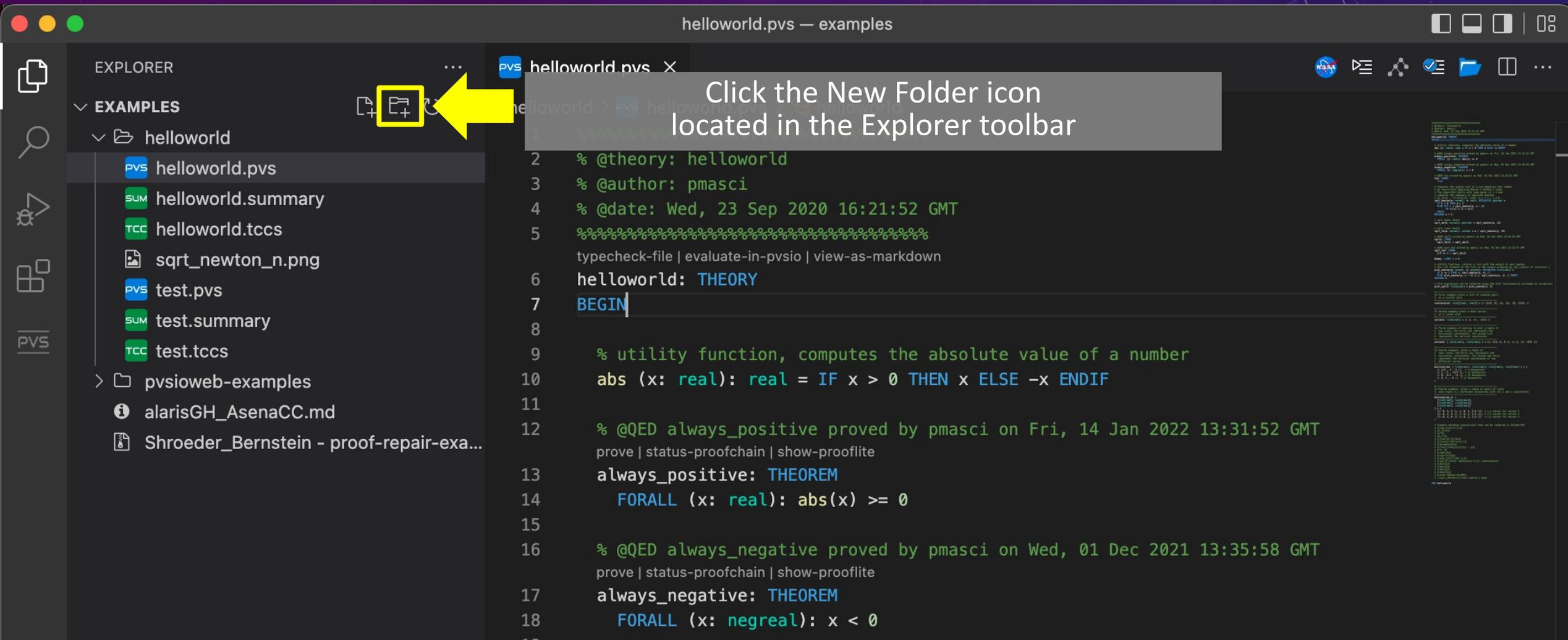
To develop a PVS theory, you need to perform 3 steps in VSCode-PVS

1. Open a workspace
2. Create a folder that will contain your .pvs files
3. Edit the .pvs files to develop the theory specification

# OPENING A WORKSPACE

The image shows a screenshot of the Visual Studio Code interface. The top bar indicates the active extension is 'PVS'. The left sidebar shows the 'EXTENSIONS: MARKETPLACE' view with a search filter 'pvs'. The main editor area displays the 'PVS v1.0.61' extension page by Paolo Masci, which includes a description, a star rating, and a 'Set File Icon Theme' button. A yellow arrow points to the 'Open Workspace...' icon in the top right corner of the editor toolbar. A grey callout box with white text says 'Click the Open Workspace icon located in the Editor toolbar'. The bottom of the page shows the extension's title 'VSCoDe-PVS: An Integrated Development Environment for the Prototype Verification System' and a 'Categories' section with 'Programming Languages' selected.

# CREATING A NEW FOLDER IN THE WORKSPACE



Click the New Folder icon located in the Explorer toolbar

```
1
2 % @theory: helloworld
3 % @author: pmasci
4 % @date: Wed, 23 Sep 2020 16:21:52 GMT
5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6 helloworld: THEORY
7 BEGIN
8
9 % utility function, computes the absolute value of a number
10 abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
11
12 % @QED always_positive proved by pmasci on Fri, 14 Jan 2022 13:31:52 GMT
13 prove | status-proofchain | show-prooflite
14 always_positive: THEOREM
15   FORALL (x: real): abs(x) >= 0
16
17 % @QED always_negative proved by pmasci on Wed, 01 Dec 2021 13:35:58 GMT
18 prove | status-proofchain | show-prooflite
19 always_negative: THEOREM
20   FORALL (x: negreal): x < 0
```

# CREATING A .PVS FILE

helloworld.pvs — examples

EXPLORER

EXAMPLES

- helloworld
  - helloworld.pvs
  - helloworld.summary
  - helloworld.tccs
  - sqrt\_newton\_n.png
  - test.pvs
  - test.summary
  - test.tccs
- pvsioweb-examples
- alarisGH\_AsenaCC.md
- Shroeder\_Bernstein - pro...

helloworld.pvs

- Open Workspace...
- New File
- New Folder
- Reveal in Finder
- Open in Integrated Terminal
- New PVS File**
- Find in Folder...
- Cut
- Copy
- Paste
- Copy Path
- Copy Relative Path
- Rename
- Delete

Right-Click on a folder and choose New PVS File

```
world
Sep 2020 16:21:52 GMT
in-pvsio | view-as-markdown
Y
ion, computes the absolute value of a number
real = IF x > 0 THEN x ELSE -x ENDIF
positive proved by pmasci on Fri, 14 Jan 2022 13:31:52 GMT
ain | show-prooflite
: THEOREM
al): abs(x) >= 0
negative proved by pmasci on Wed, 01 Dec 2021 13:35:58 GMT
ain | show-prooflite
: THEOREM
greal): x < 0
% @OED foo proved by pmasci on Wed, 01 Dec 2021 13:35:51 GMT
```

# THE CREATED .PVS FILE

```
PVS helloworld.pvs ×
helloworld > PVS helloworld.pvs > helloworld
1  %%
2  % @theory: helloworld
3  % @author: pmasci
4  % @date: Fri, 20 May 2022 17:05:04 GMT
5  %%
   typecheck-file | evaluate-in-pvsio | view-as-markdown
6  helloworld: THEORY
7  BEGIN
8  |
9  END helloworld
```

The created .pvs file will contain a documentation header and a theory declaration

# EDITING A PVS THEORY

```
typecheck-file | evaluate-in-pvsio | view-as-markdown
6  helloworld: THEORY
7  BEGIN
8
9  % utility function, computes the absolute value of a number
10 abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
11
12 % @QED always_positive proved by pmasci on Fri, 14 Jan 2022
   prove | status-proofchain | show-prooflite
13 always_positive: THEOREM
14   FORALL (x: real): abs(x) >= 0
15
```

Syntax highlighting  
for keywords, types  
and library functions

# NAVIGATING DEFINITIONS

```
typecheck-file | evaluate
6 helloworld: THEOREM
7 BEGIN
8
9 % utility function for the absolute value of a number
10 abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
11
12 % @QED always_positive proved by pmasci on Fri, 14 Jan 2022 13:31:52
13 prove | status-proofchain | show-prooflite
14 always_positive: THEOREM
15   FORALL (x: real): abs(x) >= 0
```

Builtin type `real`

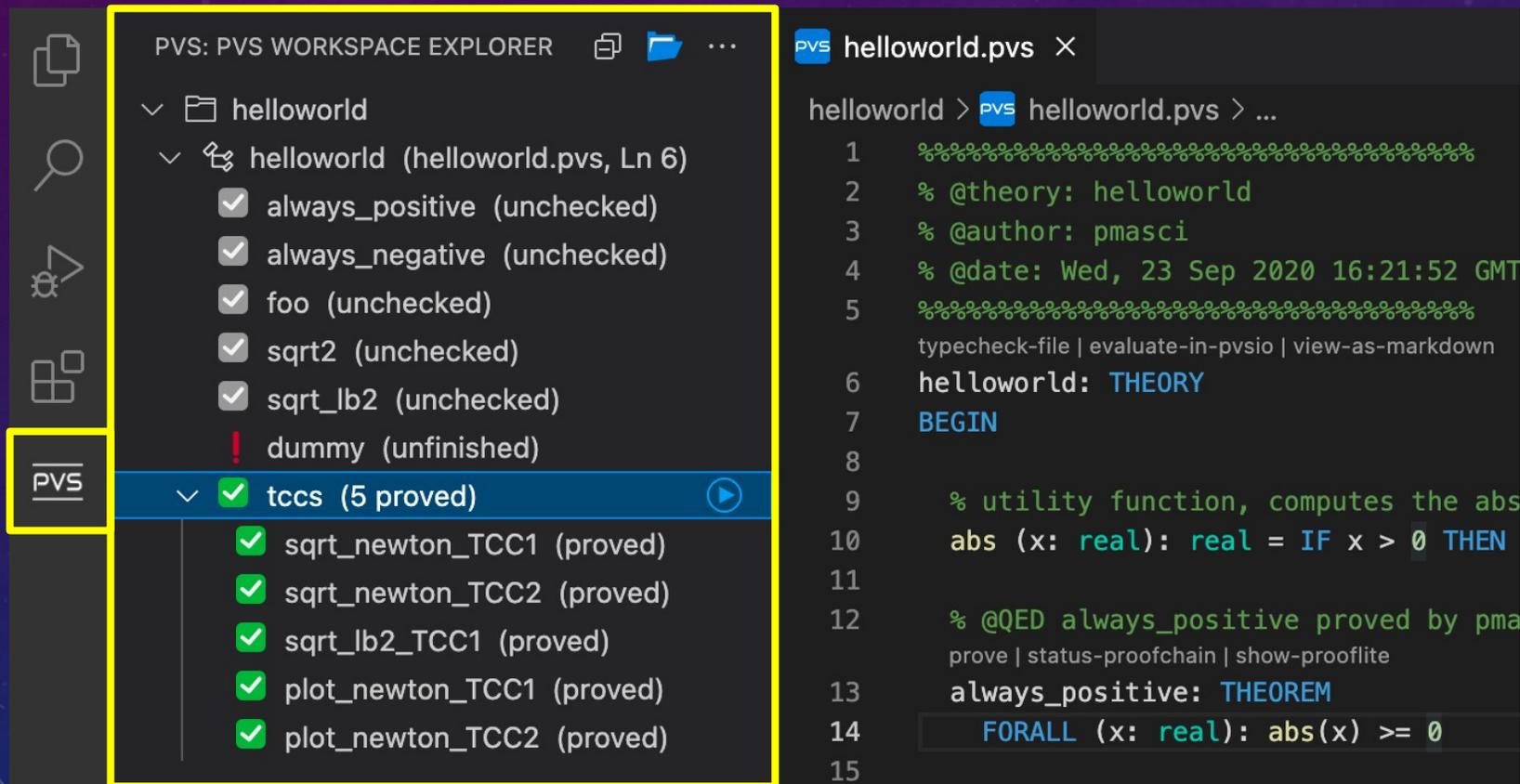
prelude (Ln 1850, Col 2)

`real`: TYPE+ FROM `number_field`

Place the mouse pointer of a term to see the definition of the term



# PROOF OBLIGATIONS (TCCS)



The screenshot shows the PVS workspace explorer on the left and the PVS editor on the right. A yellow arrow points to the PVS icon in the workspace explorer. The workspace explorer shows a tree view of the 'helloworld' project, with the 'tccs (5 proved)' folder expanded. The editor shows the PVS script for 'helloworld.pvs', which includes a theory definition and a theorem statement.

**PVS: PVS WORKSPACE EXPLORER**

- helloworld
  - helloworld (helloworld.pvs, Ln 6)
    - always\_positive (unchecked)
    - always\_negative (unchecked)
    - foo (unchecked)
    - sqrt2 (unchecked)
    - sqrt\_lb2 (unchecked)
    - dummy (unfinished)
    - tccs (5 proved)**
      - sqrt\_newton\_TCC1 (proved)
      - sqrt\_newton\_TCC2 (proved)
      - sqrt\_lb2\_TCC1 (proved)
      - plot\_newton\_TCC1 (proved)
      - plot\_newton\_TCC2 (proved)

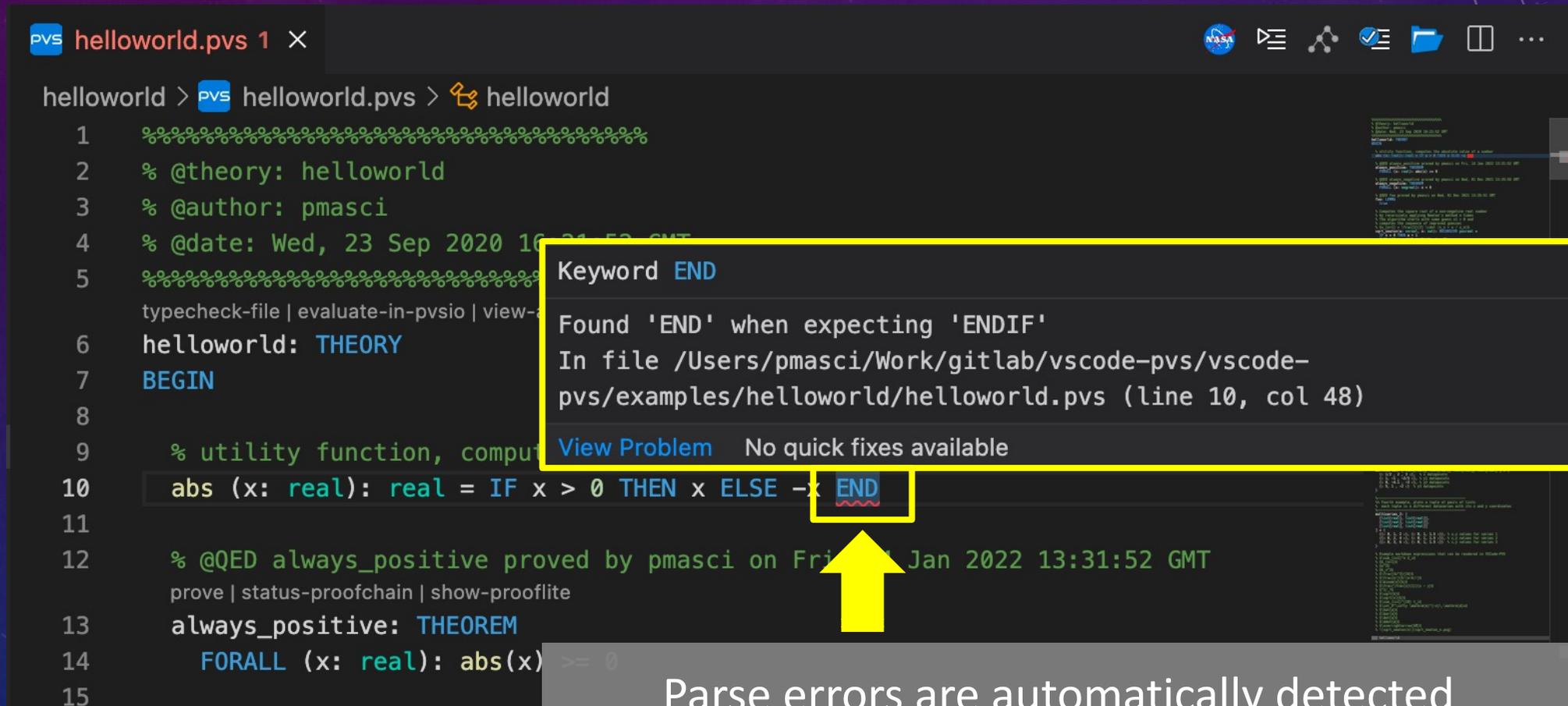
**helloworld.pvs**

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % @theory: helloworld
3  % @author: pmasci
4  % @date: Wed, 23 Sep 2020 16:21:52 GMT
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  helloworld: THEORY
7  BEGIN
8
9  % utility function, computes the absolute value of a number
10 abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
11
12 % @QED always_positive proved by pmasci on Fri, 14 Jan 2020 13:31:52 GMT
13 prove | status-proofchain | show-prooflite
14 always_positive: THEOREM
15   FORALL (x: real): abs(x) >= 0
  
```

Click the PVS icon  
to view the list of proof  
obligations and theorems

# LIVE DIAGNOSTICS FOR PARSE ERRORS



```
helloworld.pvs 1 x
helloworld > PVS helloworld.pvs > helloworld
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % @theory: helloworld
3 % @author: pmasci
4 % @date: Wed, 23 Sep 2020 16:24:52 GMT
5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6 helloworld: THEORY
7 BEGIN
8
9 % utility function, compute absolute value
10 abs (x: real): real = IF x > 0 THEN x ELSE -x END
11
12 % @QED always_positive proved by pmasci on Fri Jan 2022 13:31:52 GMT
13 prove | status-proofchain | show-prooflite
14 always_positive: THEOREM
15   FORALL (x: real): abs(x) >= 0
```

Keyword END

Found 'END' when expecting 'ENDIF'

In file /Users/pmasci/Work/gitlab/vscode-pvs/vscode-pvs/examples/helloworld/helloworld.pvs (line 10, col 48)

[View Problem](#) No quick fixes available

Parse errors are automatically detected  
when the .pvs file is saved

# LIVE DIAGNOSTICS FOR TYPECHECK ERRORS

helloworld.pvs 1 x

helloworld > pvs helloworld.pvs > helloworld

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % @theory: helloworld
3 % @author: pmasc
4 % @date: Wed, 23 Sep 2020 12:21:52 GMT
5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6 helloworld: THEORY
7 BEGIN
8
9 % utility function, computes the absolute value of a number
10 abs (x: real): string = IF x > 0 THEN x ELSE -x ENDIF
11
```

Typecheck errors are detected when you typecheck the .pvs file

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)

helloworld.pvs helloworld 1

⊗ Incompatible types for IF x > 0 THEN x ELSE -x ENDIF Typecheck error [Ln 10, Col 27] ^  
Found: number\_fields.number\_field  
Expected: strings.string

! Error: Typecheck errors in helloworld.pvs: Incompatible types fo...

# LIVE DIAGNOSTICS FOR IMPORTING ERRORS

```
helloworld.pvs 1 x
helloworld > PVS helloworld.pvs > helloworld
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % @theory: helloworld
3  % @author: pmasc
4  % @date: Wed, 23
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  helloworld: THEO
7  BEGIN IMPORTING Vector
8
9  v: Vector
```

No definition found for **Vector**  
Cannot find theory **Vector**  
Typecheck error  
View Problem Quick Fix... (⌘.)

Importing errors are detected when you typecheck the .pvs file

# QUICK-FIX ACTIONS

```
helloworld.pvs 1 x
helloworld > PVS helloworld.pvs > helloworld
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % @theory: helloworld
3  % @author: pmasc
4  % @date: Wed, 23
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  helloworld: THEO No definition found for Vector
7  BEGIN IMPORTING Vector Cannot find theory Vector
8  Typecheck error
9  v: Vector
```

No definition found for Vector  
Cannot find theory Vector  
Typecheck error  
View Problem  
Quick Fix... (⌘.)



VSCoDe-PVS provides quick-fix actions that can resolve importing errors

# QUICK-FIX ACTIONS

The screenshot shows the PVS IDE interface. The main editor displays a PVS script with the following content:

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % @theory: helloworld
3 % @author: pmasci
4 % @date: Wed, 23 Sep 2020 16:21:52 GMT
5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
typecheck-file | evaluate-in-pvsio | view-as-markdown
6 helloworld: THEORY
7 BEGIN IMPORTING Vector
8
9     v: Vector
10
11 % utility function, computes
  
```

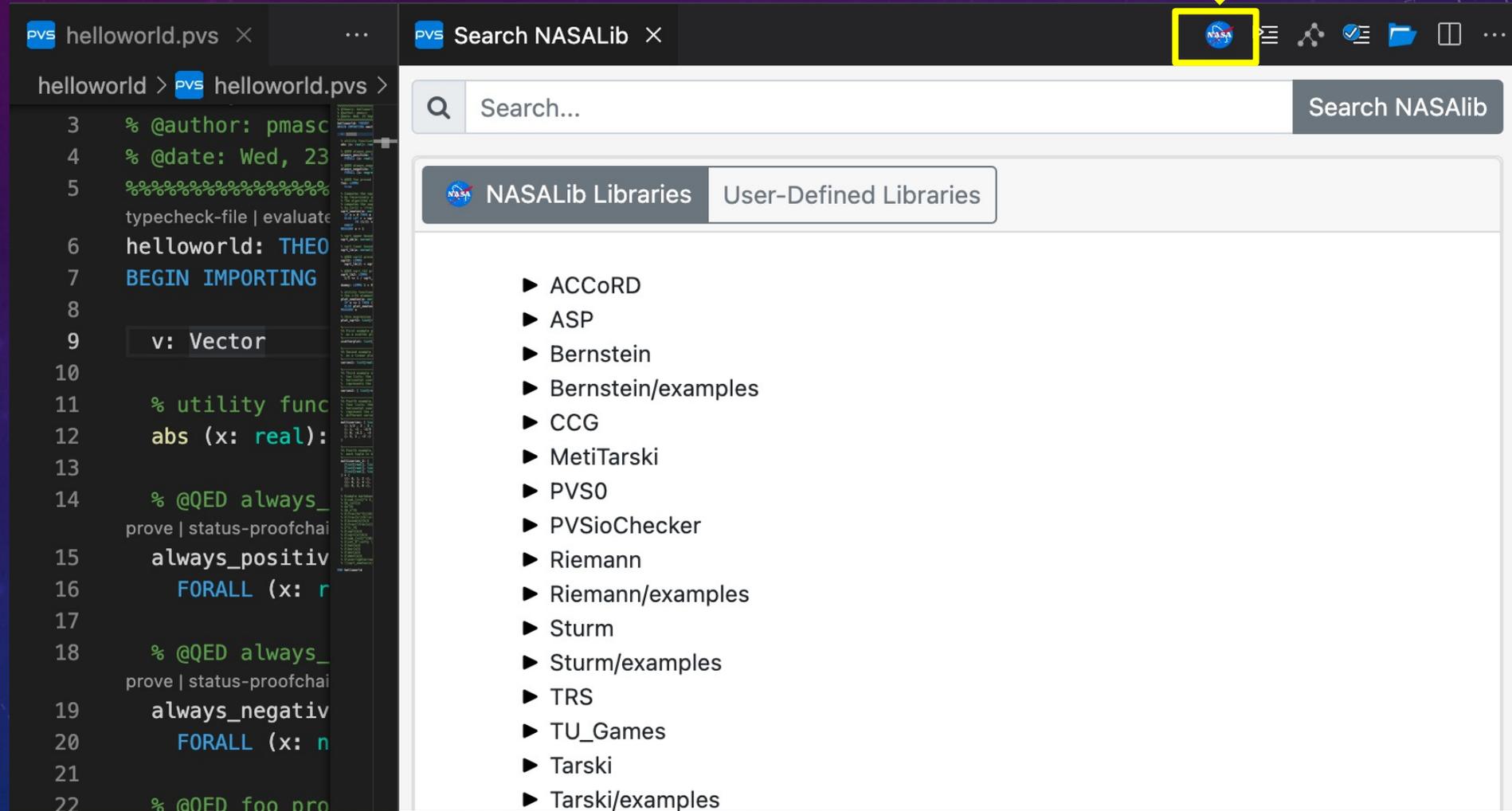
The 'PROBLEMS' panel at the bottom shows an error: 'Cannot find theory Vector Typechecked'. A yellow arrow points from the text 'Example quick-fix actions' to a list of quick-fix actions for this error. The third action, 'Change "Vector" to "vectors@vectors\_2D"', is highlighted in blue.

**Example quick-fix actions**

- Change "Vector" to "vectors@vectors\_4D"
- Change "Vector" to "vectors@vectors\_3D"
- Change "Vector" to "vectors@vectors\_2D"**
- Change "Vector" to "vectors@vectors"
- Change "Vector" to "vectors@vect3D"
- Change "Vector" to "vectors@vect2D"
- Change "Vector" to "vectors@nvectors"
- Add folder with the definition of "Vector" to PVS library path
- Open VSCode-PVS settings and edit the list of libraries in PVS library path

Click the NASA meatball logo to search definitions and lemmas in NASALib

## SEARCH NASALIB



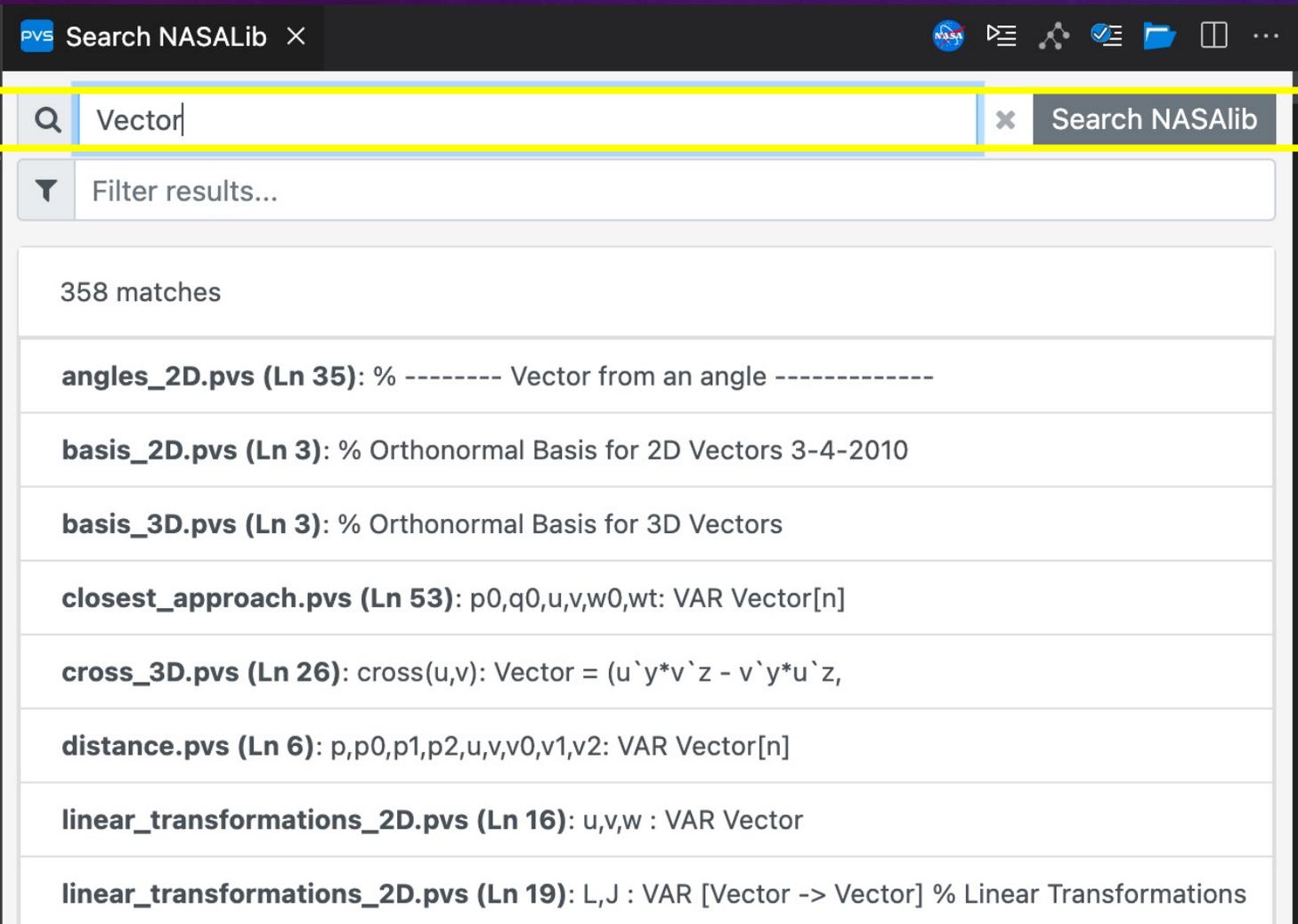
The screenshot shows the PVS IDE interface. On the left, a code editor displays the following code:

```
helloworld > PVS helloworld.pvs >
3  % @author: pmasc
4  % @date: Wed, 23
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  typecheck-file | evaluate
7  helloworld: THEO
8  BEGIN IMPORTING
9  v: Vector
10
11  % utility func
12  abs (x: real):
13
14  % @QED always_
15  prove | status-proofchai
16  always_positiv
17  FORALL (x: r
18
19  % @QED always_
20  prove | status-proofchai
21  always_negativ
22  FORALL (x: n
22  % @QED foo pro
```

On the right, the 'Search NASALib' window is open. It features a search bar with the text 'Search...' and a 'Search NASALib' button. Below the search bar, there are two tabs: 'NASALib Libraries' (selected) and 'User-Defined Libraries'. The 'NASALib Libraries' tab displays a list of libraries, each preceded by a right-pointing triangle:

- ▶ ACCoRD
- ▶ ASP
- ▶ Bernstein
- ▶ Bernstein/examples
- ▶ CCG
- ▶ MetiTarski
- ▶ PVS0
- ▶ PVSioChecker
- ▶ Riemann
- ▶ Riemann/examples
- ▶ Sturm
- ▶ Sturm/examples
- ▶ TRS
- ▶ TU\_Games
- ▶ Tarski
- ▶ Tarski/examples

# SEARCH NASALIB

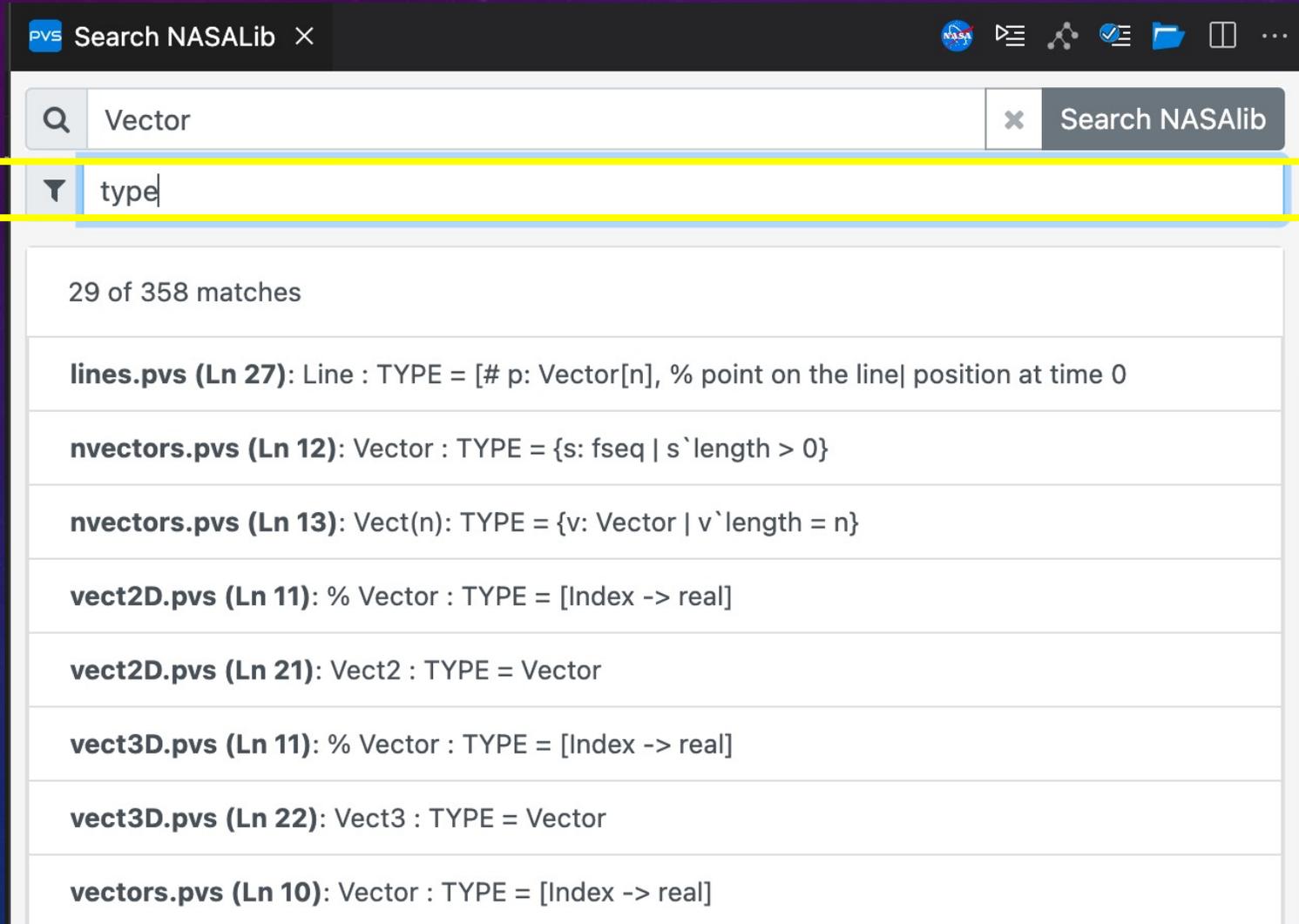


The screenshot shows a web browser window titled "Search NASALib". The search bar contains the text "Vector". A yellow arrow points to the search bar, and a yellow box highlights the search bar and the "Search NASALib" button. Below the search bar, there is a "Filter results..." dropdown menu. The search results are displayed as a list of matches, with the first match being "358 matches". The results list includes:

- angles\_2D.pvs (Ln 35):** % ----- Vector from an angle -----
- basis\_2D.pvs (Ln 3):** % Orthonormal Basis for 2D Vectors 3-4-2010
- basis\_3D.pvs (Ln 3):** % Orthonormal Basis for 3D Vectors
- closest\_approach.pvs (Ln 53):** p0,q0,u,v,w0,wt: VAR Vector[n]
- cross\_3D.pvs (Ln 26):** cross(u,v): Vector = (u`y\*v`z - v`y\*u`z,
- distance.pvs (Ln 6):** p,p0,p1,p2,u,v,v0,v1,v2: VAR Vector[n]
- linear\_transformations\_2D.pvs (Ln 16):** u,v,w : VAR Vector
- linear\_transformations\_2D.pvs (Ln 19):** L,J : VAR [Vector -> Vector] % Linear Transformations

Enter the search string in the corresponding input field and press the Search NASALib button

# SEARCH NASALIB



The screenshot shows a web browser window titled "Search NASALib". The search bar contains the text "Vector". Below the search bar, a filter dropdown menu is open, showing "type" selected. A yellow arrow points to the filter dropdown, and a yellow box highlights the filter input field. The search results are displayed below the filter, showing 29 of 358 matches. The results are listed as follows:

- lines.pvs (Ln 27):** Line : TYPE = [# p: Vector[n], % point on the line| position at time 0
- nvectors.pvs (Ln 12):** Vector : TYPE = {s: fseq | s`length > 0}
- nvectors.pvs (Ln 13):** Vect(n): TYPE = {v: Vector | v`length = n}
- vect2D.pvs (Ln 11):** % Vector : TYPE = [Index -> real]
- vect2D.pvs (Ln 21):** Vect2 : TYPE = Vector
- vect3D.pvs (Ln 11):** % Vector : TYPE = [Index -> real]
- vect3D.pvs (Ln 22):** Vect3 : TYPE = Vector
- vectors.pvs (Ln 10):** Vector : TYPE = [Index -> real]

Search results can be filtered, e.g., to show only type definitions

# PROVING A THEOREM

```
PVS helloworld.pvs ×
helloworld > PVS helloworld.pvs > helloworld

3  % @author: pmasci
4  % @date: Wed, 23 Sep 2020 16:21:52 GMT
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  typecheck-file | evaluate-in-pvsio | view-as-markdown
7  helloworld: THEORY
8  BEGIN
9
10 % utility function, computes the absolute value of a number
11 abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
12
13 % @QED always_positive proved by pmasci on Fri, 14 Jan 2022 13:31:52 GMT
14 prove | status-proofchain | show-prooflite
15 always_positive: THEOREM
16   FORALL (x: real): abs(x) >= 0
17
```

Click the inline command  
'prove' to start a proof

# PROVING A THEOREM

Proof Explorer + Proof Mate

Prover Console

The screenshot displays the PVS software interface. On the left, the 'PVS WORKSPACE EXPLORER' is visible, containing a 'PVS PROOF EXPLORER' section with a tree view showing 'always\_positive (proved)' and its sub-elements '(skosimp\*)' and '(grind)'. Below it is the 'PVS PROOF MATE' section with 'Hints' and 'Sketchpad' options. On the right, the 'Prover Console' shows the text: 'Starting prover session for always\_positive', 'always\_positive :', a proof goal '{1} FORALL (x: real): abs(x) >= 0', and a prompt '>>'. At the bottom of the console, there are instructions: '- Please enter proof command at the prover prompt / Use (help rules) to view the list of available commands' and '- Double click expands definitions. Copy / Paste text with Command+C / Command+V'. Two yellow arrows point from the text labels above to the corresponding sections in the interface.

```
Starting prover session for always_positive

always_positive :
  ┌───
  {1}  FORALL (x: real): abs(x) >= 0
  >> █

- Please enter proof command at the prover prompt / Use (help rules) to view the list of available commands
- Double click expands definitions. Copy / Paste text with Command+C / Command+V
```

# ENTERING PROOF COMMANDS

```
pvs helloworld.pvs  pvs Proving 'always_positive' ×
Starting prover session for always_positive

always_positive :
|-----
{1}  FORALL (x: real): abs(x) >= 0

>> s
keep
```

Syntax: (skeep)  
Description: Skolemize using the names of the bounded variables as the names of the skolem constants

Proof commands are entered in the Prover Console

Use TAB to autocomplete proof commands

# INTEGRATED HELP

```
pvs helloworld.pvs  pvs Proving 'always_positive' ×
```

Starting prover session for `always_positive`

```
always_positive :  
|  
{1}  FORALL (x: real): abs(x) >= 0  
  
>> s  
skeep
```

---

```
Syntax: (skeep)  
Description: Skolemize using the names of the bounded variables as the names of the skolem constants
```

Command syntax and a brief description of the command is shown at the bottom of the Prover Console

# PROOF EXPLORER

Proving 'always\_positive' — examples

PVS helloworld.pvs Proving 'always\_positive' X

PVS WORKSPACE EXPLORER

PVS PROOF EXPLORER ...

✓ always\_positive (proved)

★ (skeep)

◆ (grind)

PVS PROOF MATE ...

💡 Hints

- (assert)
- (grind)

📝 Sketchpad

📅 5/20/2022, 2:35:17 PM always\_posi...

Starting prover session for `always_positive`

```
always_positive :
|
{1}  FORALL (x: real): abs(x) >= 0
>> (skeep)

Skolemizing and keeping names of the universal formula in (+ -)

always_positive :
|
{1}  abs(x) >= 0
>> []
```

- Please enter proof command at the prover prompt / Use **(help rules)** to view the list of available comm  
 - Double click expands definitions. Copy / Paste text with Command+C / Command+V

Proof Explorer shows the  
proof tree

# PROOF EXPLORER / CONTEXT MENU

The screenshot shows the PVS Proof Explorer interface. The main window displays a proof session for 'always\_positive'. The left sidebar shows the 'PVS WORKSPACE EXPLORER' with a tree view containing 'PVS PROOF EXPLORER' and 'always\_positive (proved)'. A context menu is open over a node in the tree, listing actions such as 'Run subtree', 'Fast forward here', 'Rewind here', 'Jump here', 'Show sequent', 'Add new proof command', 'Copy', 'Cut', 'Paste', 'Copy subtree', and 'Cut subtree'. A yellow arrow points to the 'Fast forward here' option. The main editor shows the proof state with the command 'FORALL (x: real): abs(x) >= 0' and the goal 'abs(x) >= 0'.

Point-and-click actions can be used for undo/redo, fast-forward, re-play

Edit and copy/paste operations are restricted

# PROOF MATE

Proving 'always\_positive' — examples

PVS helloworld.pvs PVS Proving 'always\_positive' X

PVS WORKSPACE EXPLORER

PVS PROOF EXPLORER ✓ ◀ || ▶ ⋮

✓ always\_positive (proved)

★ (skip)

◆ (grind) ✎

PVS PROOF MATE || ▶ ⋮

💡 Hints

- (assert)
- (grind)

📄 Sketchpad 🗑️ + 🔄

📅 5/20/2022, 2:35:17 PM always\_posi...

◆ (skosimp\*)

Starting prover session for `always_positive`

`always_positive :`

$$\text{FORALL } (x: \text{real}): \text{abs}(x) \geq 0$$

`>> (skip)`

Skolemizing and keeping names of the universal formula in (

`always_positive :`

$$\text{abs}(x) \geq 0$$

`>> []`

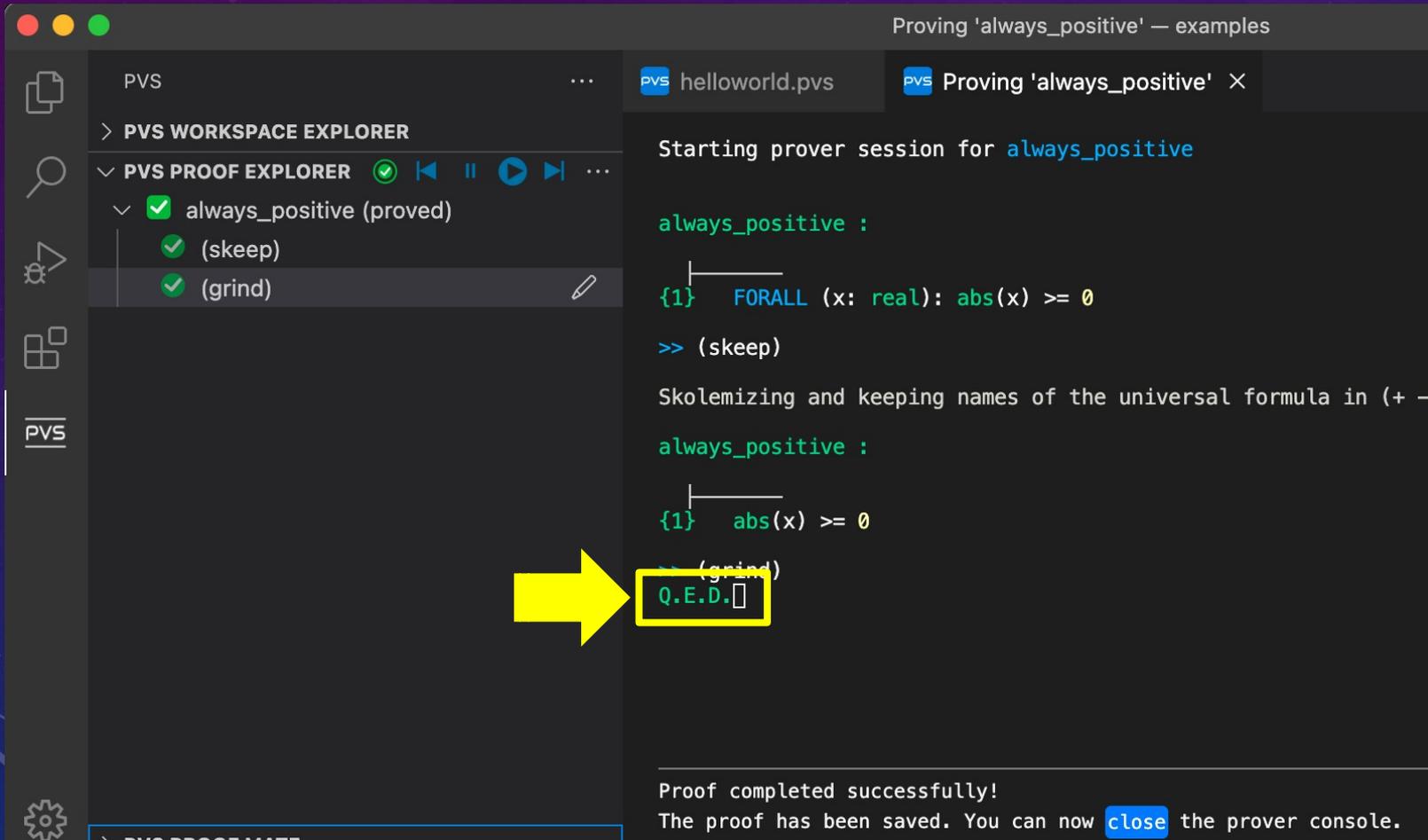
– Please enter proof command at the prover prompt / Use **(help rules)** to view the list of available comm

– Double click expands definitions. Copy / Paste text with **Command+C** / **Command+V**

Proof Mate supports  
unrestricted editing and  
copy/paste

Hints suggest proof  
commands

# PROOF COMPLETE!



```
Proving 'always_positive' — examples
PVS helloworld.pvs Proving 'always_positive' X
PVS WORKSPACE EXPLORER
PVS PROOF EXPLORER
  always_positive (proved)
    (skip)
    (grind)
Starting prover session for always_positive
always_positive :
  ┌───
  {1}  FORALL (x: real): abs(x) >= 0
  >> (skip)
Skolemizing and keeping names of the universal formula in (+ -)
always_positive :
  ┌───
  {1}  abs(x) >= 0
  >> (grind)
  Q.E.D.
Proof completed successfully!
The proof has been saved. You can now close the prover console.
```

A message 'Q.E.D.' in the Prover Console indicates proof complete

A @QED comment is automatically added to the corresponding theorem in the .pvs file

# DOCUMENTING YOUR PVS FILES

VSCoDe-PVS provides functionalities to support documentation of theories

- @QED annotation for proved theorems
- Header annotations for new theories
- Pretty-printing of comments written in the markdown language

# USING MARKDOWN SYNTAX IN THE COMMENTS

```

typecheck-file | evaluate-in-pvsio | view-as-markdown
6  helloworld: THEORY
7  BEGIN
8
9  % utility function, computes the absolute value of a number
10 abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
11
12 % @QED always_positive proved by pmasci on Fri, 14 Jan 2022 13:31:5
   prove | status-proofchain | show-prooflite
13 always_positive: THEOREM
14   FORALL (x: real): abs(x) >= 0
15
16   % Example markdown expressions that can be used in the documentatio
17   %  $\sum_{i=1}^n X_i$ 
18   %  $k_{n+1}$ 
19   %  $n^2$ 
20   %  $k_n^2$ 
21   %  $\frac{4z^3}{16}$ 
22   %  $\frac{n!}{k!(n-k)!}$ 
23   %  $\binom{n}{k}$ 
24   %  $\frac{\frac{x}{1}}{x - y}$ 
25   %  $3/7$ 
26   %  $\sqrt{k}$ 
27   %  $\sqrt[n]{k}$ 
28   %  $\sum_{i=1}^{10} t_i$ 
29   %  $\int_0^{\infty} \mathrm{e}^{-x} \mathrm{d}x$ 
30   %  $\hat{a}$ 
31   %  $\bar{a}$ 
32   %  $\dot{a}$ 
33   %  $\ddot{a}$ 
34   %  $\overrightarrow{AB}$ 
   open(sqrt_newton(n)) | view-as-markdown
35   % ![sqrt newton(n)](sqrt newton n.png)
36
37 END helloworld

```

Example markdown expressions that can be used in the theory:

- Math equations
- Inline links
- Figures

# MARKDOWN SYNTAX / LIVE PREVIEW

```

typecheck-file | evaluate-in-pvs | view-as-markdown
6 helloworld: THEORY
7 BEGIN
8
9 % utility function, computes the absolute value of a number
10 abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
11
12 % @QED always_positive proved by pmasci on Fri, 14 Jan 2022 13:31:52 GMT
13 always_positive: THEOREM
14 FORALL (x: real): abs(x) >= 0
15
16 % Example markdown expressions that can be used in the documentatio
17
18 % $k_{n+1}$
19 % $n^2$
20 % $k_n^2$
21 % $\frac{4z^3}{16}$
22 % $\frac{n!}{k!(n-k)!}$
23 % $\binom{n}{k}$
24 % $\frac{\frac{x}{y}}{3/7}$
25 % $\sqrt{k}$
26 % $\sqrt[n]{k}$
27 % $\sum_{i=1}^{10} t_i$
28 % $\int_0^{\infty} e^{-x} dx$
29 % $\hat{a}$
30 % $\bar{a}$
31 % $\acute{a}$
32 % $\ddot{a}$
33 % $\overrightarrow{AB}$
34
35 open(sqrt_newton(n)) | view-as-markdown
36 % ![sqrt_newton(n)](sqrt_newton_n.png)

```

```

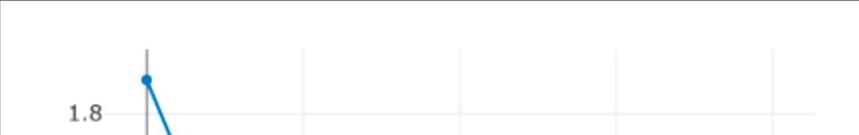
helloworld: THEORY
BEGIN

% utility function, computes the absolute value of a number
abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF

% @QED always_positive proved by pmasci on Fri, 14 Jan 2022 13:31:52 GMT
always_positive: THEOREM
FORALL (x: real): abs(x) >= 0

% Example markdown expressions that can be used in the documentatio
% $\sum_{i=1}^n X_i$
% $k_{n+1}$
% $n^2$
% $k_n^2$
% $\frac{4z^3}{16}$
% $\frac{n!}{k!(n-k)!}$
% $\binom{n}{k}$
% $\frac{x}{y}$
% $3/7$
% $\sqrt{k}$
% $\sqrt[n]{k}$
% $\sum_{i=1}^{10} t_i$
% $\int_0^{\infty} e^{-x} dx$
% $\hat{a}$
% $\bar{a}$
% $\acute{a}$
% $\ddot{a}$
% $\overrightarrow{AB}$

```

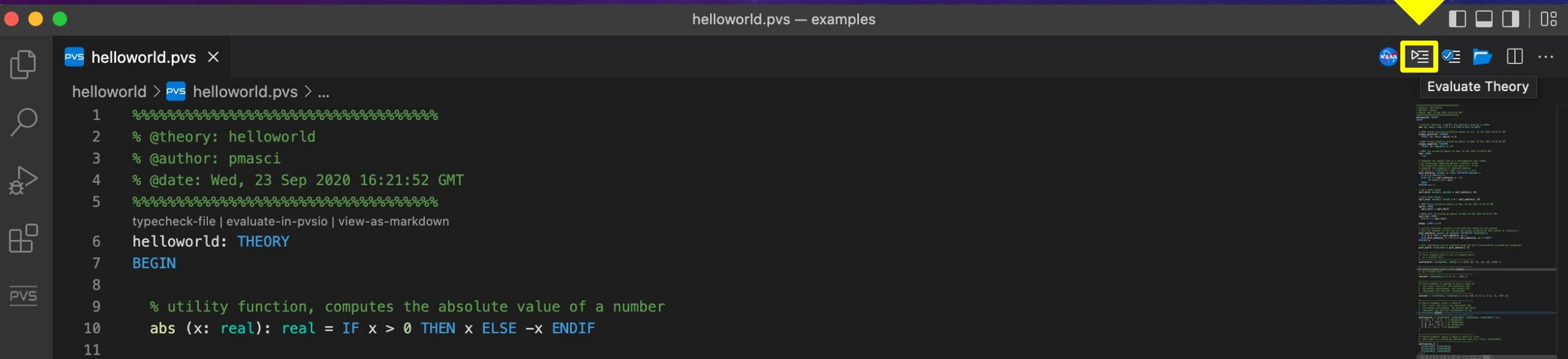




Click 'view-as-markdown' to view the pretty-printed version

# EVALUATING PVS EXPRESSIONS

Click the 'play' icon in the editor toolbar



The screenshot shows the PVS editor interface. The main editor window displays the following PVS theory code:

```
helloworld > pvs helloworld.pvs > ...
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % @theory: helloworld
3 % @author: pmasci
4 % @date: Wed, 23 Sep 2020 16:21:52 GMT
5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6 typecheck-file | evaluate-in-pvsio | view-as-markdown
7 helloworld: THEORY
8 BEGIN
9 % utility function, computes the absolute value of a number
10 abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
11
```

The editor toolbar at the top right contains several icons. A yellow arrow points to the 'play' icon (a square with a right-pointing triangle), which is used to evaluate the theory. To the right of the editor, the 'Evaluate Theory' panel is visible, showing the results of the evaluation.

# EVALUATING PVS EXPRESSIONS

```

helloworld.pvs x
helloworld > pvs helloworld.pvs > ...
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % @theory: helloworld
3  % @author: pmasci
4  % @date: Wed, 23 Sep 2020 16:21:52 GMT
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  typecheck-file | evaluate-in-pvsio | view-as-markdown
7  helloworld: THEORY
8  BEGIN
9  % utility function, computes the absolute value of a number
10 % abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
11

```

Enter ground expressions at the evaluator prompt

```

... PVS Evaluating helloworld x
Starting PVSio evaluator session for theory helloworld

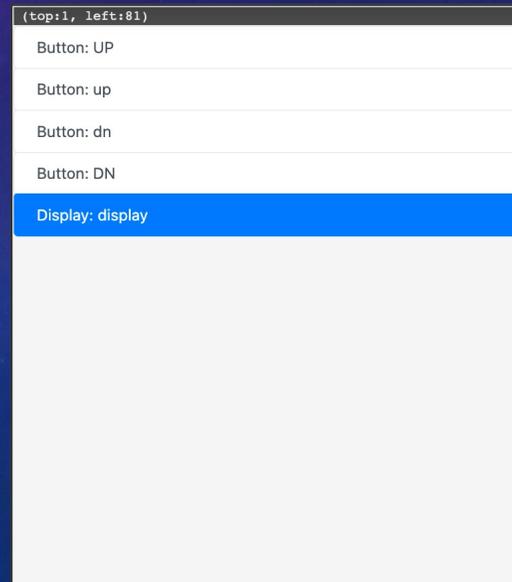
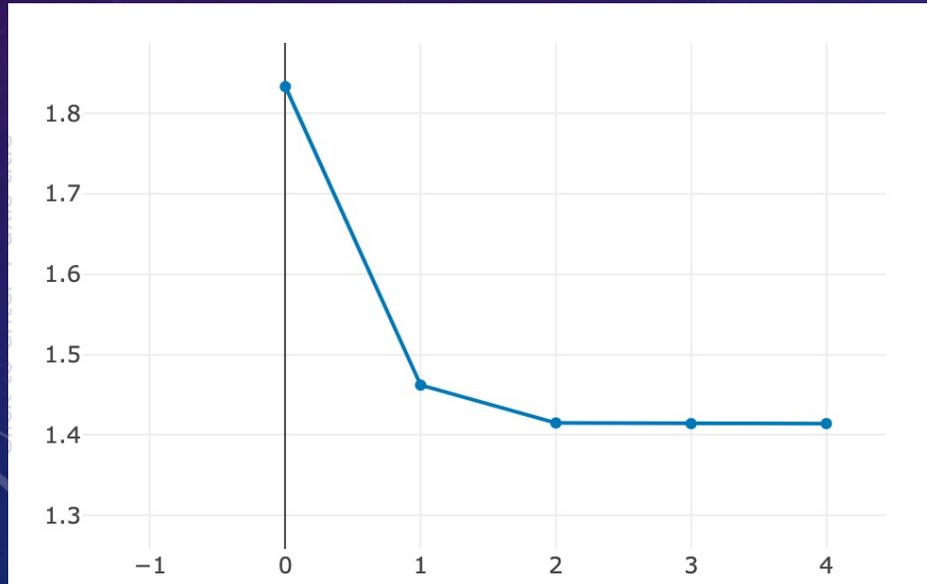
PVSio Evaluator
Usage:
- Enter a PVS expression followed by ';'
  or
- Enter a Lisp expression followed by '!'

<PVSio> abs(-3);
==>
3

```

# PLOTTING + RAPID PROTOTYPING

VSCoDe-PVS extends the evaluation mechanism of PVS with front-ends for plotting functions and creating interactive prototypes



# PLOTTING FUNCTIONS

```
helloworld > pvs helloworld.pvs > helloworld
```

```
22 true
23
24 % Computes the square root of a non-negative real number
25 % by recursively applying Newton's method n times
26 % The algorithm starts with some guess x1 > 0 and
27 % computes the sequence of improved guesses
28 %  $x_{n+1} = \frac{1}{2} \cdot (x_n + a / x_n)$ 
29 sqrt_newton(a: nreal, n: nat): RECURSIVE posreal =
30     IF n = 0 THEN a + 1
31     ELSE LET r = sqrt_newton(a, n - 1)
32         IN (1/2) * (r + a/r)
33     ENDIF
34 MEASURE n + 1
35
36 % utility function, returns a list with the output of sqrt_newton
37 % the i-th element of the list is the output produced by sqrt_newton
38 plot_newton(a: nreal, n: posnat): RECURSIVE list[real] =
39     IF n <= 1 THEN (: sqrt_newton(a, n) :)
40     ELSE plot_newton(a, n - 1) o (: sqrt_newton(a, n) :) ENDIF
41 MEASURE n
42
43 % this expression can be rendered using the plot functionality plot
44 plot-expression
45 plot_sqrt2: list[real] = plot_newton(2, 5)
```

Define a function, e.g., 'sqrt\_newton'

# PLOTTING FUNCTIONS

```

helloworld > pvs helloworld.pvs > helloworld
22 true
23
24 % Computes the square root of a non-negative real number
25 % by recursively applying Newton's method n times
26 % The algorithm starts with some guess x1 > 0 and
27 % computes the sequence of improved guesses
28 %  $x_{n+1} = \frac{1}{2} \cdot (x_n + a / x_n)$ 
29 sqrt_newton(a: nreal, n: nat): RECURSIVE posreal =
30   IF n = 0 THEN a + 1
31   ELSE LET r = sqrt_newton(a, n - 1)
32     IN (1/2) * (r + a/r)
33   ENDIF
34 MEASURE n + 1
35
36 % utility function, returns a list with the output of sqrt_newton
37 % the i-th element of the list is the output produced by sqrt_newton
38 plot_newton(a: nreal, n: posnat): RECURSIVE list[real] =
39   IF n <= 1 THEN (: sqrt_newton(a, n) :)
40   ELSE plot_newton(a, n - 1) o (: sqrt_newton(a, n) :) ENDIF
41 MEASURE n
42
43 % this expression can be rendered using the plot functionality p
44 plot-expression
45 plot_sqrt2: list[real] = plot_newton(2, 5)

```

Create a function that evaluates `sqrt_newton` and stores the results into a list of reals

# PLOTTING FUNCTIONS

```

helloworld >  helloworld.pvs >  helloworld
22
23
24 % Computes the square root of a non-negative real number
25 % by recursively applying Newton's method n times
26 % The algorithm starts with some guess x1 > 0 and
27 % computes the sequence of improved guesses
28 %  $x_{n+1} = \frac{1}{2} \cdot (x_n + a / x_n)$ 
29 sqrt_newton(a: nreal, n: nat): RECURSIVE posreal =
30   IF n = 0 THEN a + 1
31   ELSE LET r = sqrt_newton(a, n - 1)
32     IN (1/2) * (r + a/r)
33   ENDIF
34 MEASURE n + 1
35
36
37
38
39
40 ELSE plot_newton(a, n - 1) o (: sqrt_newton(a, n) :) ENDIF
41 MEASURE n
42
43 % this expression can be rendered using the plot functionality p
44 plot_sqrt2: list[real] = plot_newton(2, 5)
45
46
47
48 %-----

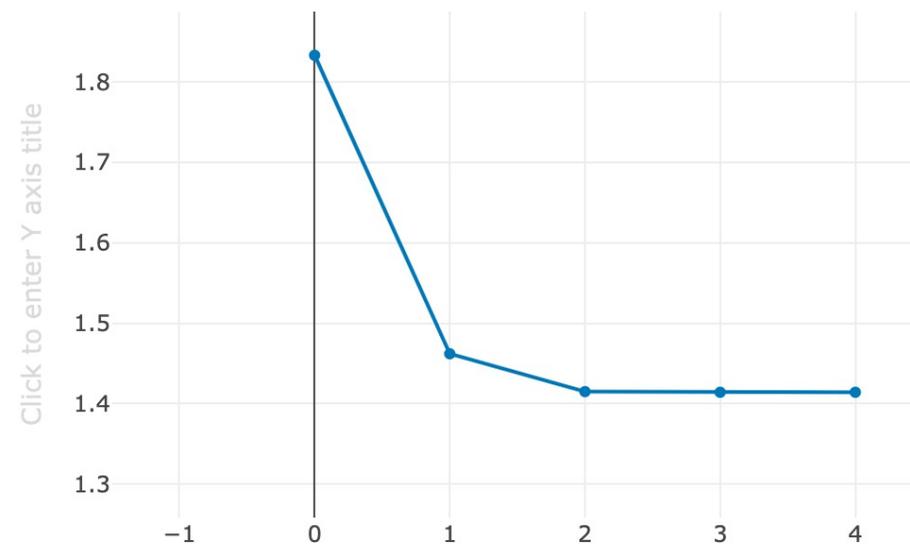
```

Plot the saved list of reals by clicking the inline command 'plot-expression'

(: 11/6, 193/132, 72097/50952, 10390190017/7346972688, 215912063945802350977/152672884556058511392 :)

Linear
  SemiLog
  LogLog

plot\_sqrt2



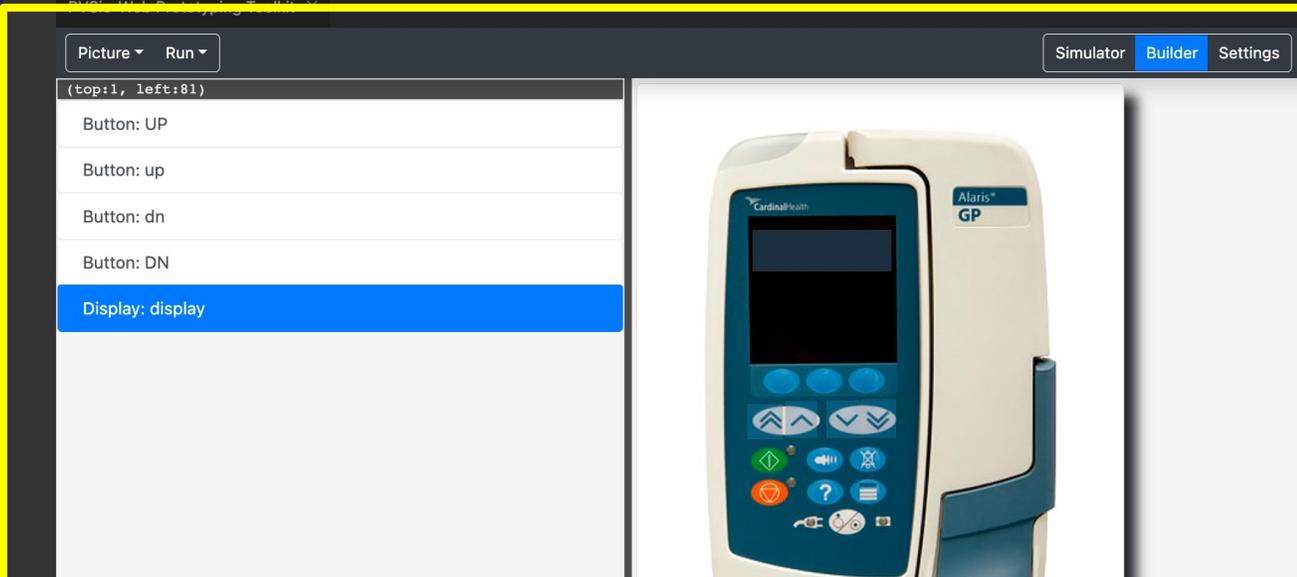
# CREATING INTERACTIVE PROTOTYPES

```

pvs alarisGP.pvs x
pvsioweb-examples > dataEntry > AlarisGP > pvs alarisGP.pvs > alarisGP
33
34 state: TYPE = [# display: alaris_real, timer: alaris_timer, step: alaris_step, power_led: ledType #]
35
36 init(x: alaris_real): state = (# display := x, timer := max_timer, step := small_step, power_led := OFF #)
37
38
39
40 %-- utility functions
41 trim(x: real): alaris_real = IF x > max THEN max ELSIF x < 0 THEN 0 ELSE x ENDIF
42 ceil(x: real): real = ceiling(x)
43
44
45
46 %-- alaris' chevron (UP,up,dn,DN)
47 alaris_up(delta: alaris_step, val: alaris_real): alaris_real =
48   IF val < 100 THEN trim( floor((val*10) + delta) / 10 )
49   ELSIF val >= 100 AND val < 1000 THEN trim( floor((val) + delta) )
50   ELSE trim( (floor(val/10) + delta) * 10 ) ENDIF
51

```

Click the 'play' button in the editor toolbar and select PVSio-web from the menu that will be displayed



A new panel will be opened where you can build and simulate a prototype driven by the PVS theory

Examples: <https://github.com/pvsioweb/examples/>

# COMMAND SHORTCUTS

Commands can be triggered with keyboard shortcuts initiated with the sequence M-x, where M is the META key (Alt on Linux, Option (⌘) on Mac)

## Frequent Commands

M-x tc *(typecheck file)*  
M-x tcp *(typecheck file and re-run all proofs)*  
M-x show-tccs *(show proof obligations)*  
M-x parse *(parse file)*  
M-x pr *(prove formula at the cursor location)*  
M-x prt *(prove theory, i.e., re-run all proofs)*  
M-x pri *(prove importchain, i.e., re-run all proofs including those in imported theories)*  
M-x pvsio *(start PVSio)*  
M-x x-show-proof *(shows proof tree)*  
M-x show-proof-summary *(show proof summary)*  
M-x show-prooflite *(show prooflite script)*  
M-x insert-prooflite-script *(insert prooflite script at cursor location)*  
M-x status-proof-chain *(status proof chain)*  
M-x vpf *(view prelude file)*

## Additional Commands

M-x add-pvs-library *(add a folder to the vscode-pvs library path)*  
M-x pvs-library-path *(show pvs library)*  
M-x reset-pvs-library-path *(resets the vscode-pvs library path to empty)*  
M-x reboot-pvs *(reboot pvs-server)*  
M-x clean-bin *(remove pvsbin files created by pvs)*  
M-x clean-tccs *(remove .tccs files created by pvs)*  
M-x clean-all *(remove temporary files, including .tccs and pvsbin)*  
M-x install-pvs *(install or update PVS)*  
M-x install-nasalib *(install NASALib)*  
M-x update-nasalib *(update the installed version of NASALib)*  
M-x set-pvs-path *(sets the path to the PVS executables)*  
M-x settings *(shows vscode-pvs settings)*  
M-x welcome *(shows vscode-pvs welcome screen)*

# KEY RESOURCES

## Examples presented in this tutorial

- <https://github.com/nasa/vscode-pvs/tree/master/vscode-pvs/examples>
- <https://github.com/pvsioweb/examples>

## Tools

- VSCode-PVS (source code, user manual, tutorials): <https://github.com/nasa/vscode-pvs>
- PVS Allegro (pvs language reference, documentation): <https://pvs.csl.sri.com/>
- Visual Studio Code (user interface guide): <https://code.visualstudio.com/docs/getstarted/userinterface>
- PVS Google Group: <https://groups.google.com/g/pvs-group>

## Publications

- Paolo Masci and César Muñoz, [An Integrated Development Environment for the Prototype Verification System](#), F-IDE Workshop, Electronic Proceedings in Theoretical Computer Science (EPTCS), Vol. 310, pp. 35-49, 2019
- Paolo Masci and Aaron Dutle, [Proof Mate: an Interactive Proof Helper for PVS](#), NASA Formal Methods Symposium (NFM2022), Lecture Notes in Computer Science, Springer, 2022 (to appear)