

TUTORIAL ON VS CODE-PVS

PAOLO MASI

PAOLO.MASCI@NIANET.ORG

NATIONAL INSTITUTE OF AEROSPACE
LANGLEY RESEARCH CENTER

MAY 2022

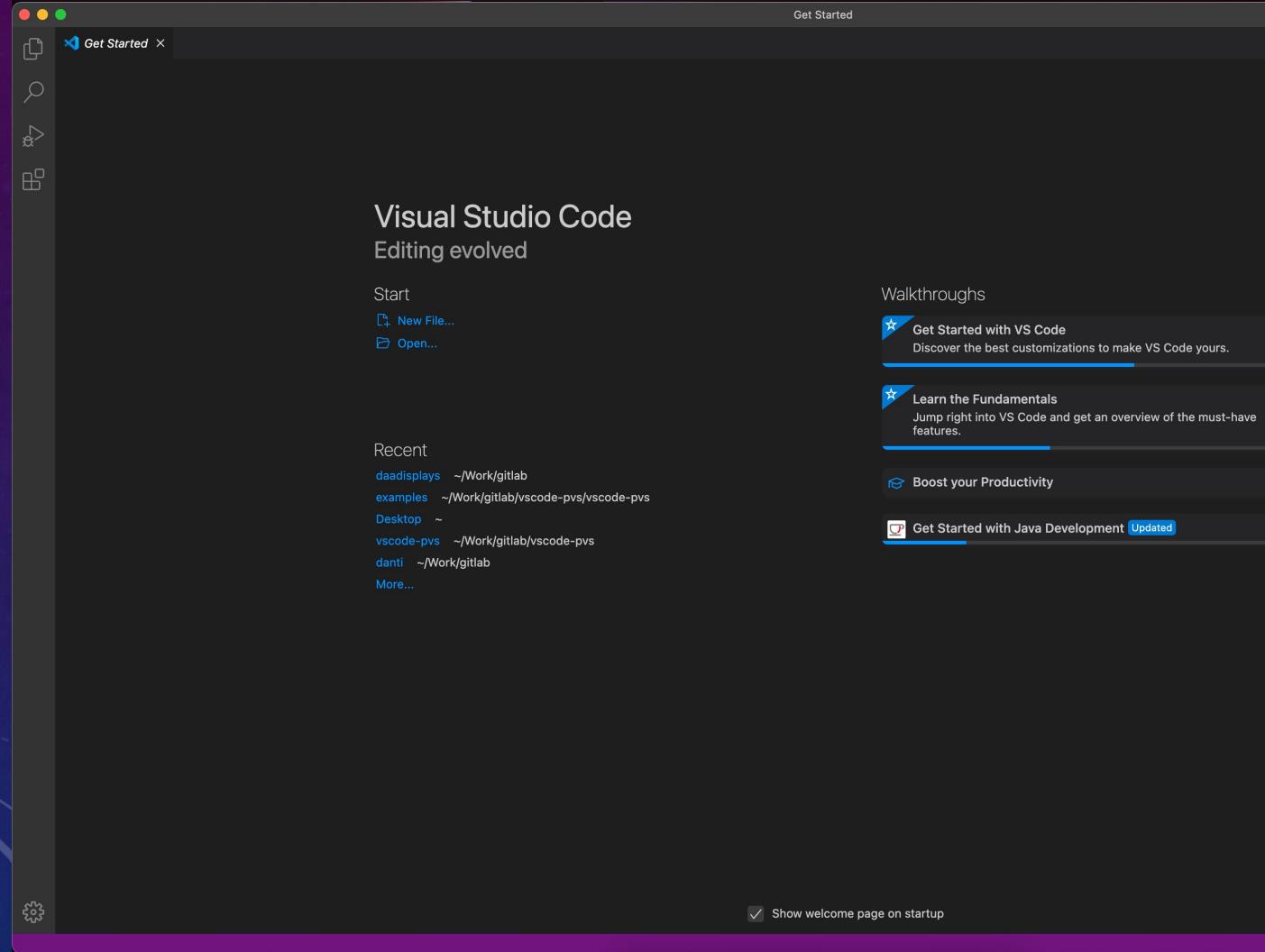
TOPICS COVERED IN THIS TUTORIAL

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 VS CODE-PVS

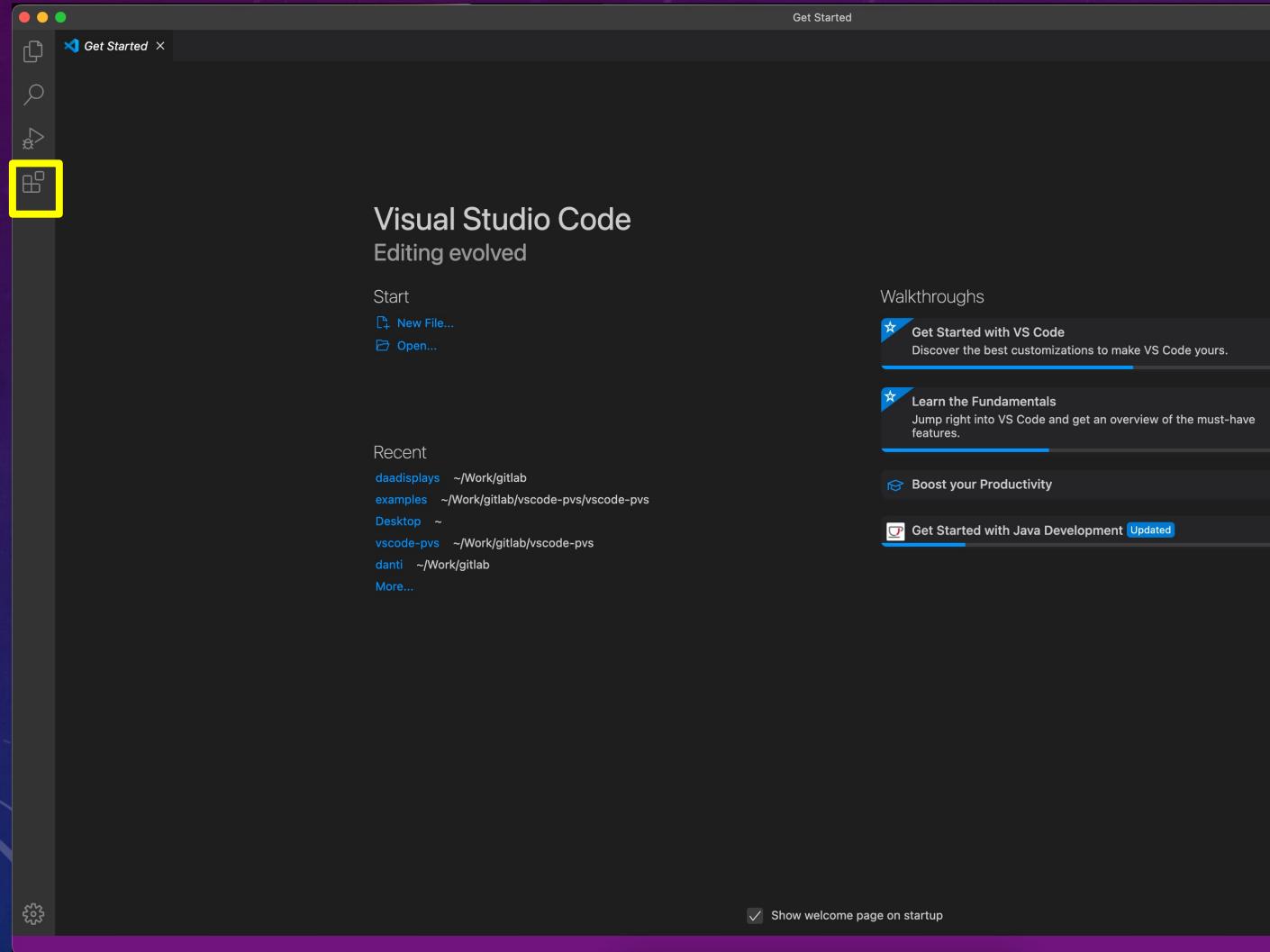
- VSCode-PVS can be installed from the Visual Studio Code marketplace
- Requirements
 - Visual Studio Code (<https://code.visualstudio.com/download>)
 - NodeJS (<https://nodejs.org/en/download/>)
 - Linux or Intel Mac

INSTALLATION OF VS CODE-PVS



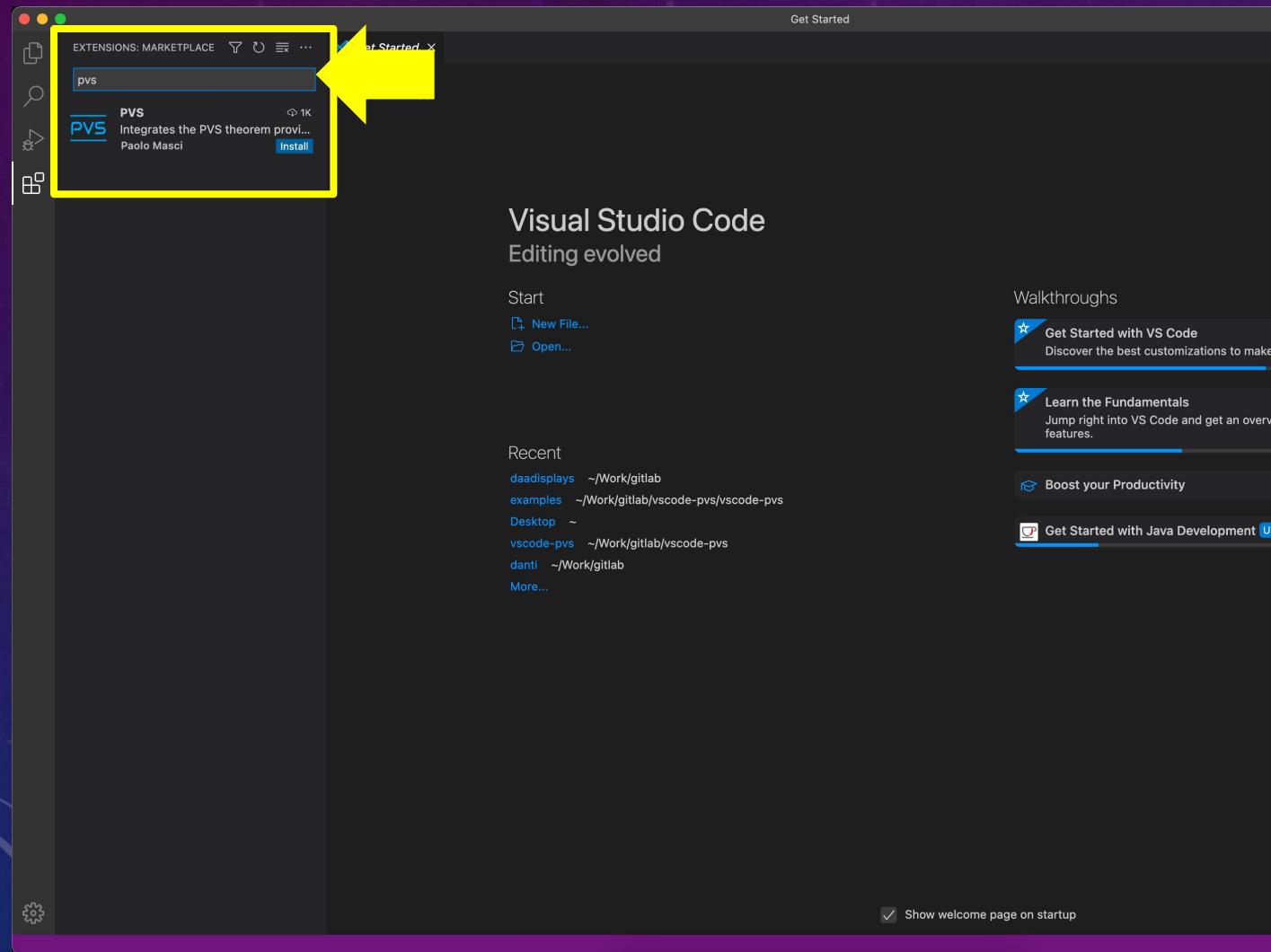
1. Open Visual Studio Code

INSTALLATION OF VS CODE-PVS



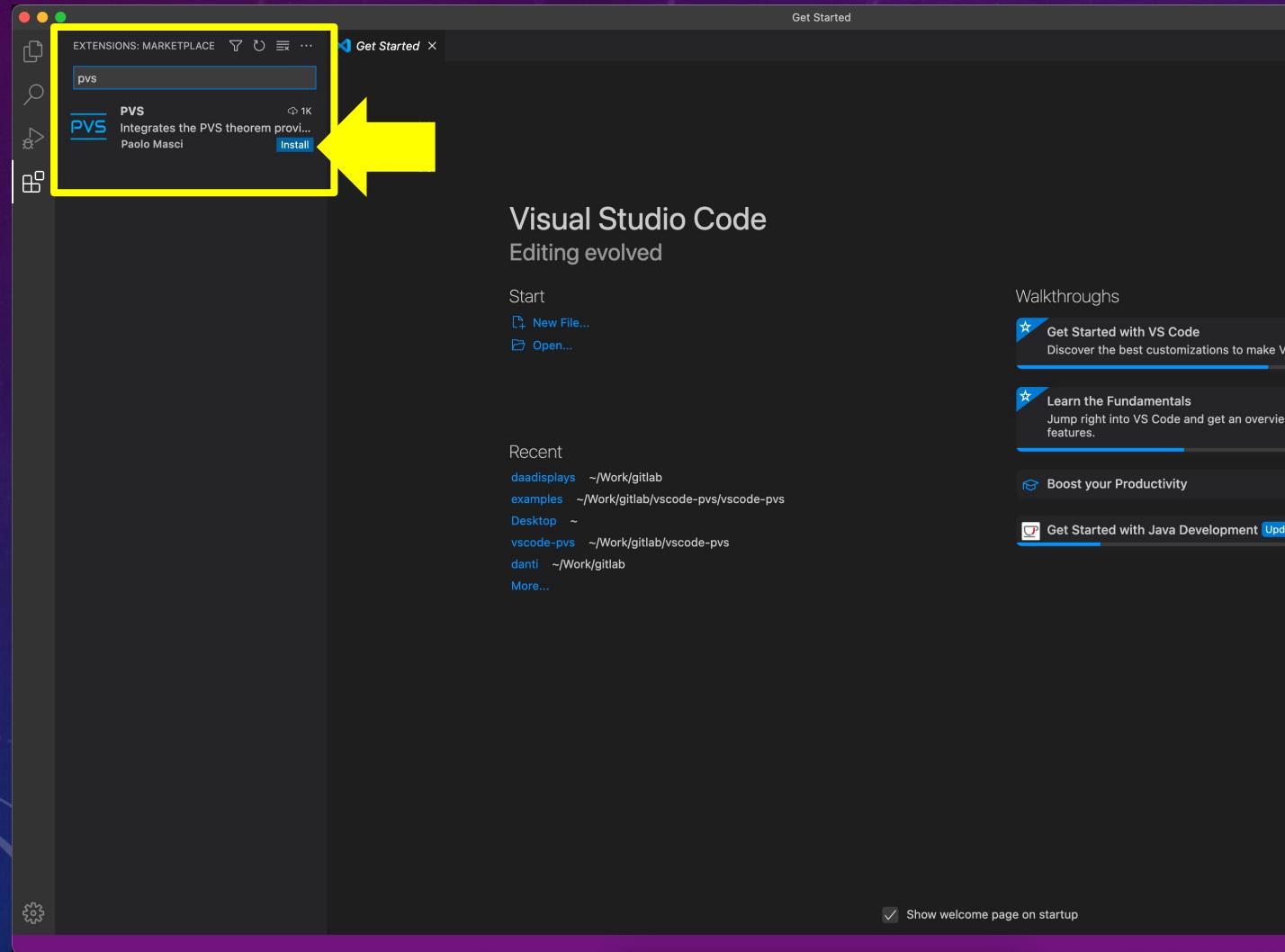
1. Open Visual Studio Code
2. Click on the Extensions icon

INSTALLATION OF VS CODE-PVS



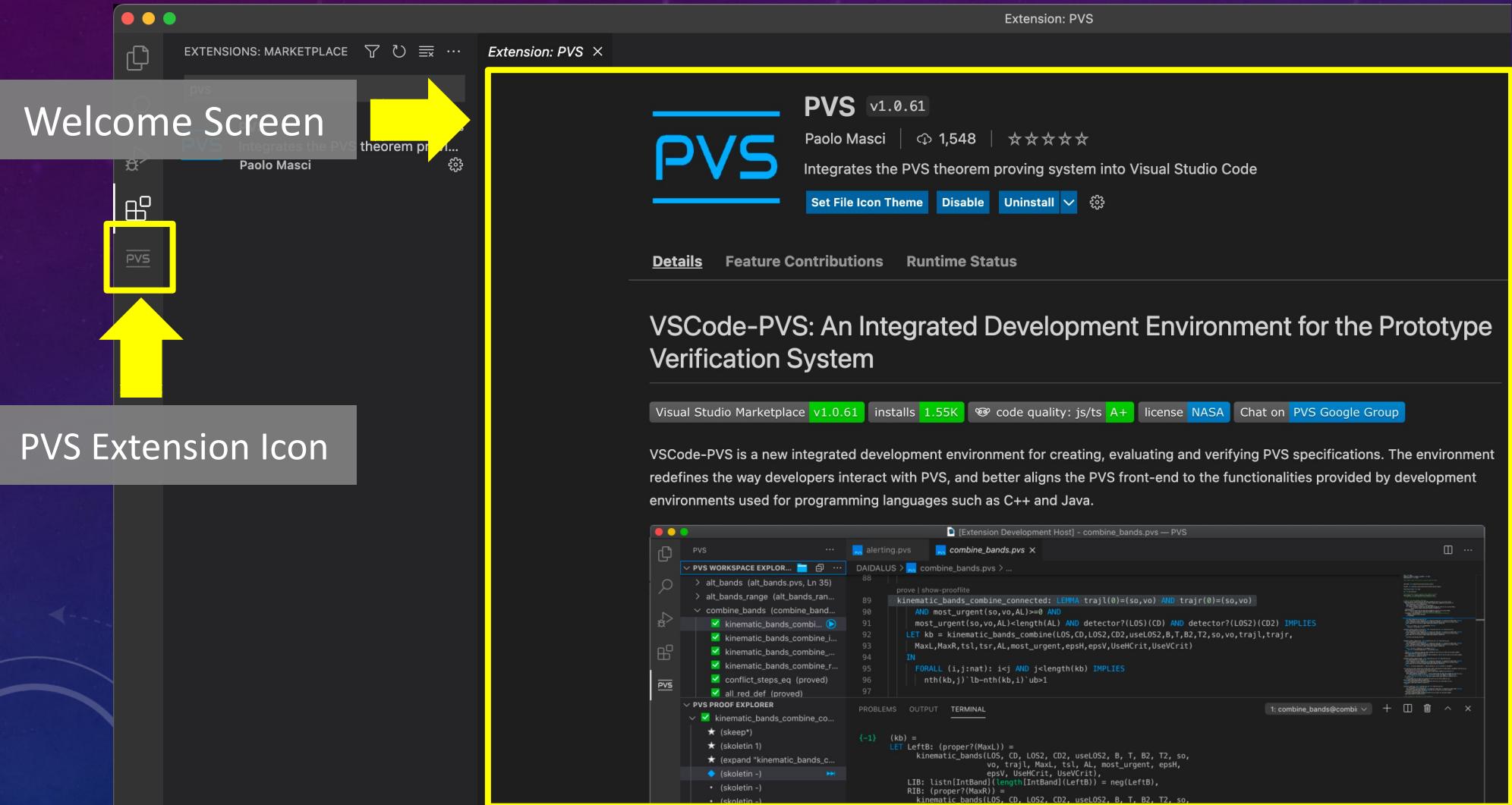
1. Open Visual Studio Code
2. Click on the Extensions icon
3. Search PVS in the Marketplace

INSTALLATION OF VS CODE-PVS



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

INSTALLATION OF VS CODE-PVS



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

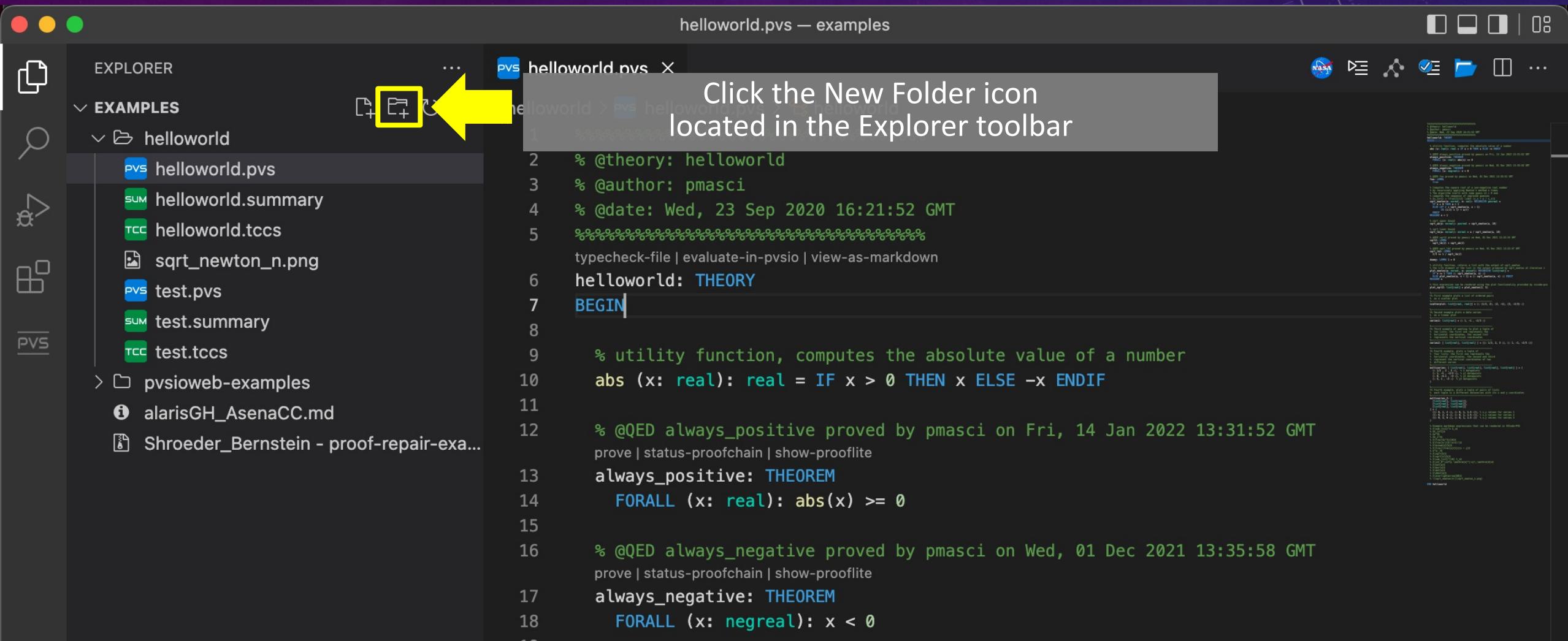
1. Open a workspace
2. Create a folder that will contain your .pvs files
3. Edit the .pvs files to develop the theory specification

OPENING A WORKSPACE

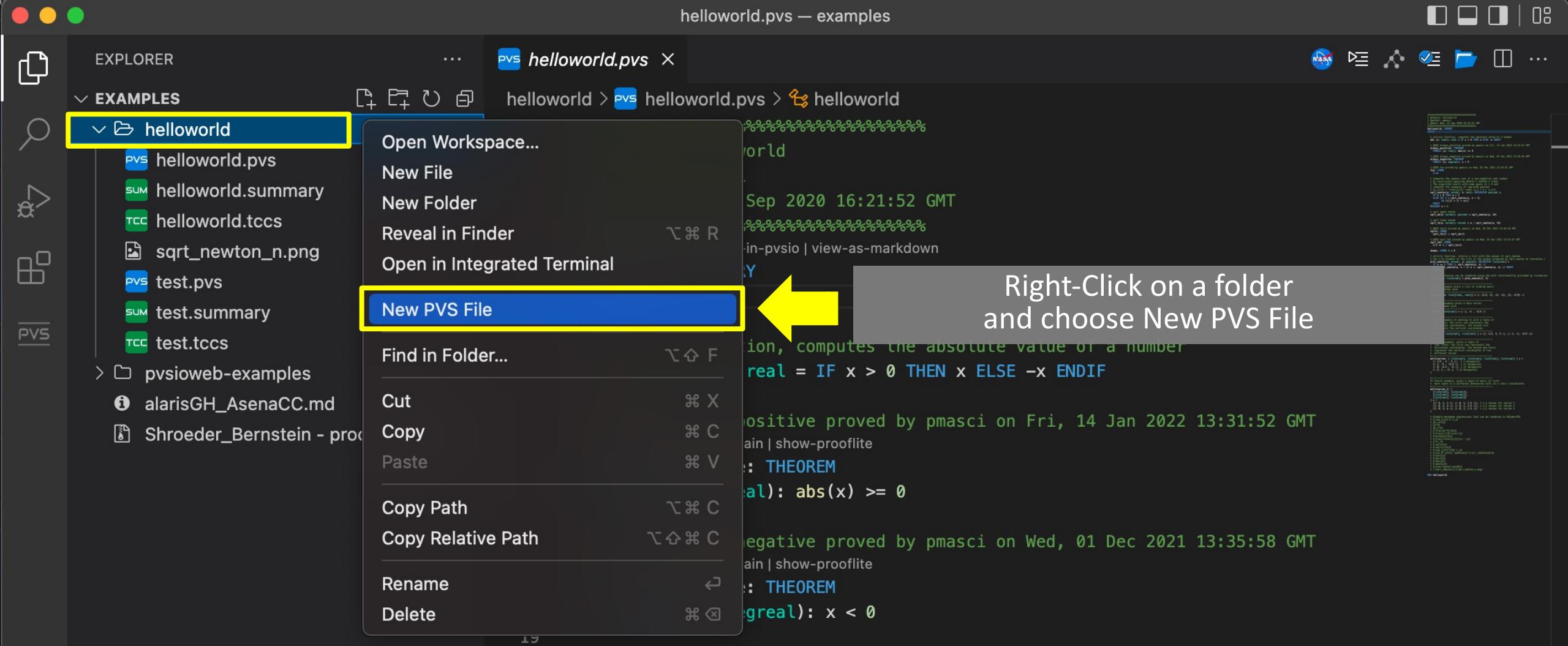
The screenshot shows the VSCode interface with the following elements:

- Left Sidebar:** Shows icons for File, Find, Go, and others.
- Top Bar:** Shows "EXTENSIONS: MARKETPLACE" and a search bar containing "pvs".
- Central Area:** Displays the "Extension: PVS" details page.
 - PVS Logo:** Large blue "PVS" logo.
 - Extension Information:** Name: PVS, Version: v1.0.61, Author: Paolo Masci, Published: 240ms ago, Downloads: 1,548, Rating: ★★★★☆.
 - Description:** Integrates the PVS theorem prover.
 - Status:** Enabled globally.
 - Toolbar:** Includes "Set File Icon Theme", "Disable", "Uninstall", and other options.
 - Text Overlay:** "Click the Open Workspace icon located in the Editor toolbar" with a yellow arrow pointing to the "Open Workspace..." button in the toolbar.
- Right Sidebar:** Shows "Categories" (Programming Languages) and "Resources".

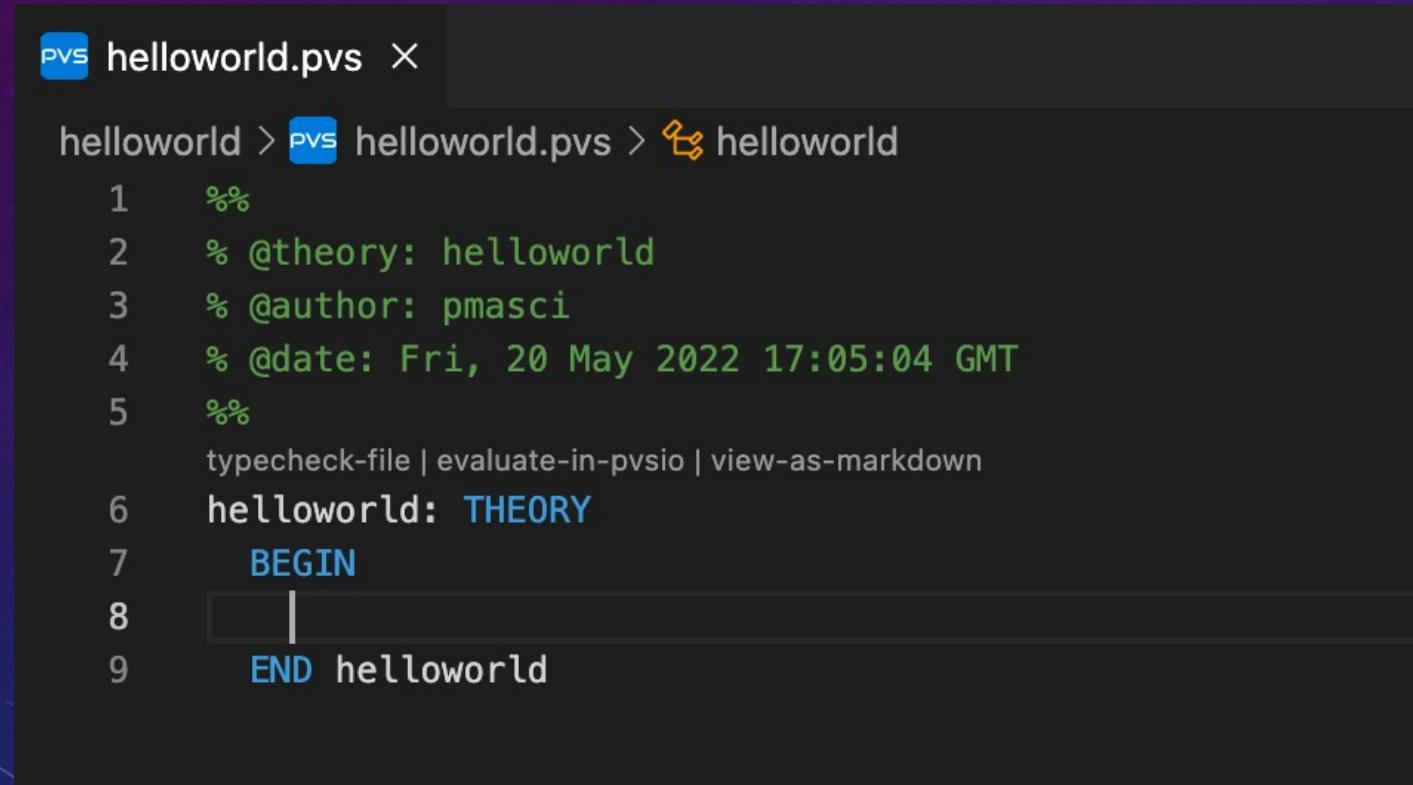
CREATING A NEW FOLDER IN THE WORKSPACE



CREATING A .PVS FILE



THE CREATED .PVS FILE



The screenshot shows a PVS editor window with the file "helloworld.pvs" open. The file contains the following code:

```
1  %%  
2  % @theory: helloworld  
3  % @author: pmasci  
4  % @date: Fri, 20 May 2022 17:05:04 GMT  
5  %%  
6  typecheck-file | evaluate-in-pvsi | view-as-markdown  
7  helloworld: THEORY  
8  BEGIN  
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
13     prove | status-proofchain | show-prooflite
14     always_positive: THEOREM
15         FORALL (x: real): abs(x) >= 0
```

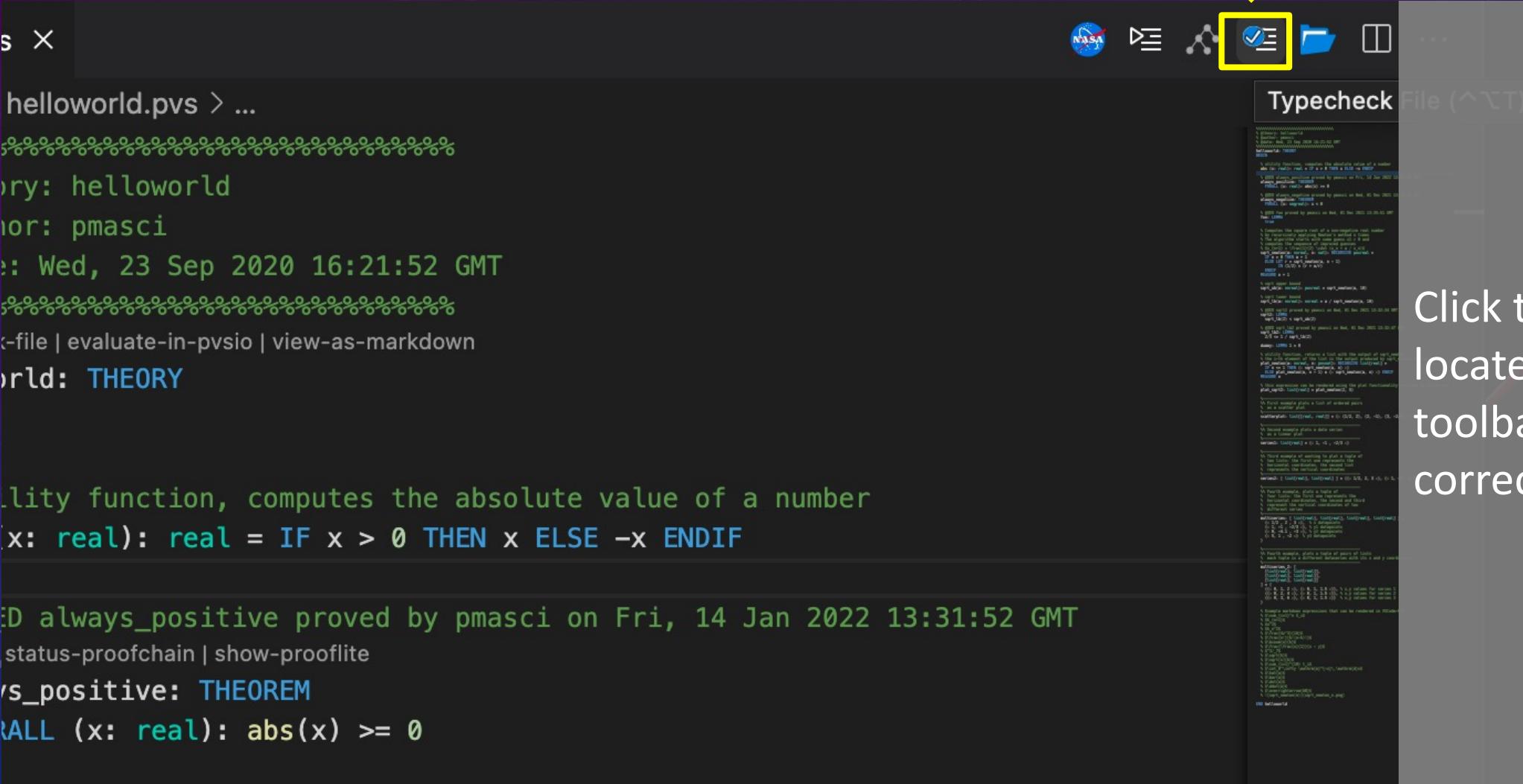
Syntax highlighting
for keywords, types
and library functions

NAVIGATING DEFINITIONS

```
typecheck-file | evaluate
6 helloworld: THEOREM
7 BEGIN
8
9   % utility function
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
```

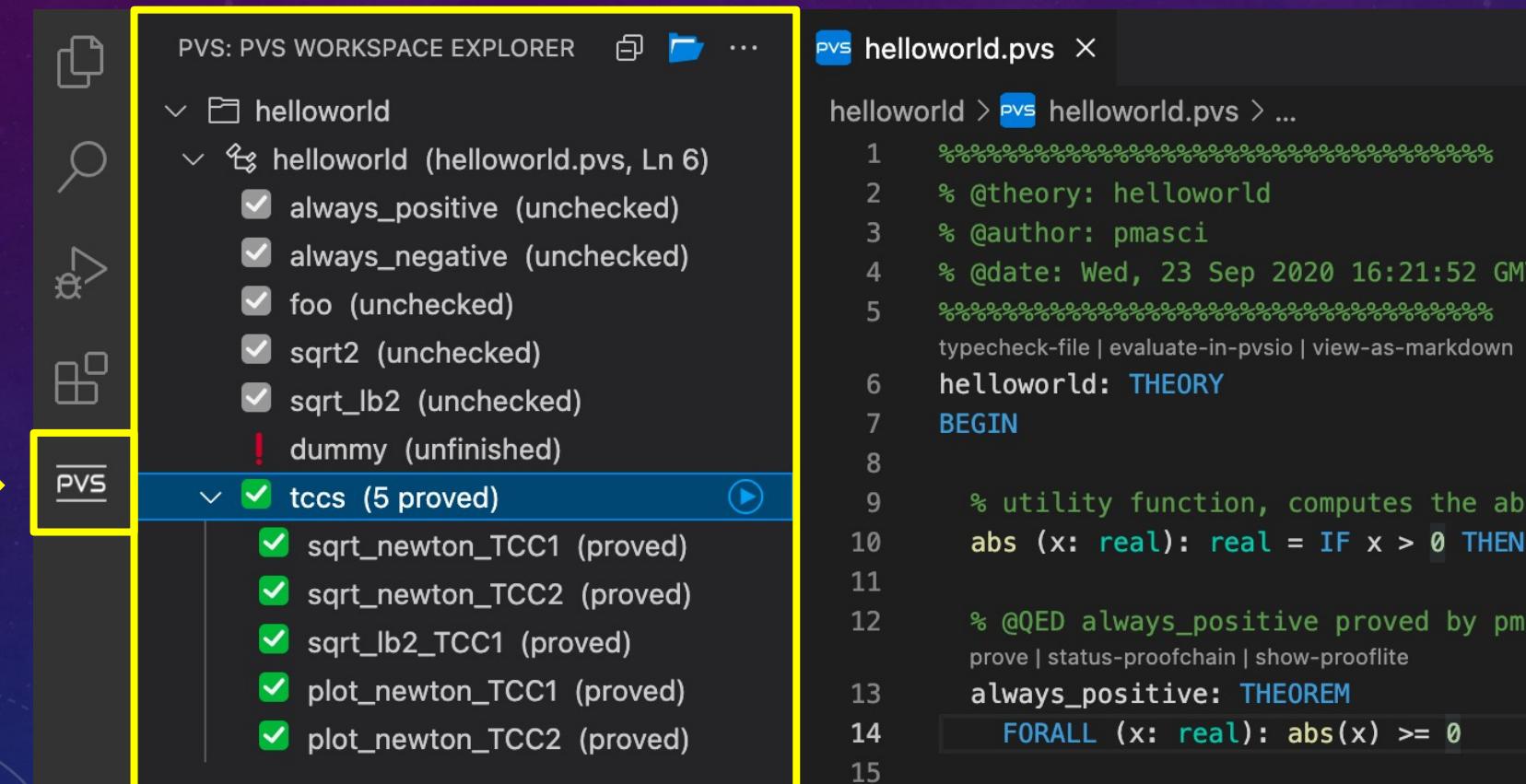
Place the mouse pointer of a term to see the definition of the term

CHECKING TYPE CORRECTNESS



Click the Typecheck icon located in the Editor toolbar to check type correctness

PROOF OBLIGATIONS (TCCS)



The screenshot shows the PVS workspace explorer on the left and a code editor on the right. A yellow arrow points from the PVS icon in the workspace explorer to the code editor. The workspace explorer lists several proof obligations and theorems under the 'tccs' category, which is highlighted with a yellow border. The code editor displays the source code for 'helloworld.pvs'.

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)

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

LIVE DIAGNOSTICS FOR PARSE ERRORS

The screenshot shows the VS Code interface with a dark theme. A file named `helloworld.pvs` is open in the editor. The code contains a parse error at line 10, column 48. A yellow box highlights the error message, which reads:

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)

A yellow arrow points from the text "Parse errors are automatically detected when the .pvs file is saved" to the word "END" in the error message.

```
1  %%%
2  % @theory: helloworld
3  % @author: pmasci
4  % @date: Wed, 23 Sep 2020 16:21:52 GMT
5  %%%
6  typecheck-file | evaluate-in-pvso | view-
7  helloworld: THEORY
8  BEGIN
9  % utility function, comput
10 abs (x: real): real = IF x > 0 THEN x ELSE -x END
11
12 % @QED always_positive proved by pmasci on Friday, 21 Jan 2022 13:31:52 GMT
13 prove | status-proofchain | show-prooflite
14 always_positive: THEOREM
15 FORALL (x: real): abs(x) >= 0
```

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

LIVE DIAGNOSTICS FOR TYPECHECK ERRORS

The screenshot shows a code editor interface with a dark theme. At the top, there's a toolbar with icons for NASA, file operations, and more. Below the toolbar, the file path is shown as `helloworld > pvs helloworld.pvs > helloworld`. The main editor area contains the following PVS-Studio code:

```
1  %%%%%%
2  % @theory: helloworld
3  % @author: pmasci
4  % @date: Wed, 23 Sep 2020 17:21:52 GMT
5  %%%%%%
6  typecheck-file | evaluate-in-pvsi | view-as-markdown
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
```

A yellow arrow points from the text "Typecheck errors are detected when you typecheck the .pvs file" down to the line `abs (x: real): string = IF x > 0 THEN x ELSE -x ENDIF`, which is highlighted with a yellow box.

In the bottom right corner of the editor, there's a small preview window showing the output of the typecheck command.

At the bottom of the screen, the VS Code status bar shows tabs for PROBLEMS (1), OUTPUT, DEBUG CONSOLE, and TERMINAL. The PROBLEMS tab is active, displaying a single error:

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, **/*.ts, **/node_modules/**) X

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

The screenshot shows a PVS editor interface with the file `helloworld.pvs` open. The code is as follows:

```
1 %%%%%%
2 % @theory: helloworld
3 % @author: pmasc
4 % @date: Wed, 23
5 %%%%%%
6 typecheck-file | evaluate
7 helloworld: THEORY
8 BEGIN IMPORTING Vector
9 v: Vector
```

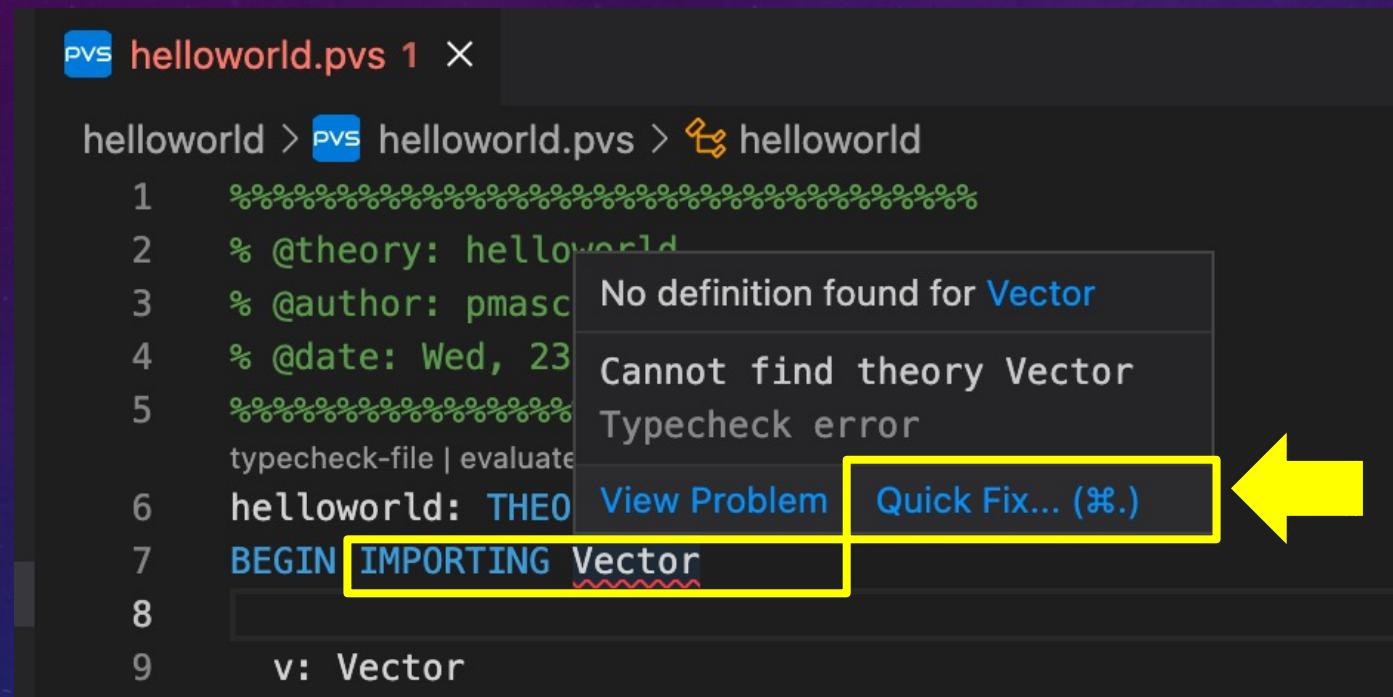
A yellow box highlights the error message for line 7, which is "BEGIN IMPORTING Vector". The message is:

No definition found for Vector
Cannot find theory Vector
Typecheck error

Below the error message are two buttons: "View Problem" and "Quick Fix... (⌘.)". A yellow arrow points from the text "Importing errors are detected when you typecheck the .pvs file" to the "View Problem" button.

Importing errors are detected when you typecheck the .pvs file

QUICK-FIX ACTIONS



helloworld.pvs 1 ×

helloworld > helloworld.pvs > helloworld

```
1 %%%%%%
2 % @theory: helloworld
3 % @author: pmasc
4 % @date: Wed, 23
5 %%%%%%
6 typecheck-file | evaluate
7 helloworld: THEO
8 BEGIN IMPORTING Vector
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 a VSCode interface with a dark theme. The top bar includes icons for NASA, file operations, and settings. The main area displays a PVS file named `helloworld.pvs`. The code contains a `Vector` import statement. A yellow arrow points from the text "Example quick-fix actions" to a floating menu of nine suggestions:

- Change "Vector" to "vectors@vectors_4D"
- Change "Vector" to "vectors@vectors_3D"
- Change "Vector" to "vectors@vectors_2D"** (highlighted in blue)
- Change "Vector" to "vectors@vectors"
- Change "Vector" to "vectors@vect3D"
- Change "Vector" to "vectors@vect2D"
- Change "Vector" to "vectors@nvector"
- 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

The bottom status bar shows "PROBLEMS 1", "OUTPUT", and "DEBUG CONSOLE". A problem icon indicates one issue: "Cannot find theory Vector Typechecked".

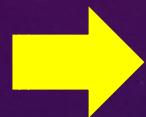
Click the NASA meatball logo to search definitions and lemmas in NASALib

SEARCH NASALIB

The screenshot shows a PVS (Practical Verifier for Systems) interface with two windows open:

- helloworld.pvs**: This window displays a PVS script with code related to vectors and real numbers, including annotations like `% @author: pmasc`, `% @date: Wed, 23`, and `% @QED always_`. The code includes functions like `abs` and `FORALL` statements.
- Search NASALib**: This window is a library browser. It features a search bar at the top with the placeholder "Search...". Below the search bar are tabs for "NASALib Libraries" and "User-Defined Libraries". The "NASALib Libraries" tab is selected, showing a list of library names:
 - ACCoRD
 - ASP
 - Bernstein
 - Bernstein/examples
 - CCG
 - MetiTarski
 - PVS0
 - PVSSoChecker
 - Riemann
 - Riemann/examples
 - Sturm
 - Sturm/examples
 - TRS
 - TU_Games
 - Tarski
 - Tarski/examples

SEARCH NASALIB



Screenshot of the PVS Search NASALib interface. A yellow box highlights the search input field, and a yellow arrow points to it from the left.

The search input field contains the text "Vector".

The search results section displays 358 matches:

- 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

PVS Search NASALib X

Vector

type

29 of 358 matches

- 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-pvsi | 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
12  % @OED 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
```

Click the inline command
'prove' to start a proof

PROVING A THEOREM

Proof Explorer + Proof Mate

Prover Console

The screenshot shows the PVS (Primer Verification System) IDE interface. At the top, there are two tabs: "Proof Explorer + Proof Mate" on the left and "Prover Console" on the right. Below these tabs is a dark-themed code editor window.

PVS WORKSPACE EXPLORER:

- PVS PROOF EXPLORER:** Shows a proof named "always_positive" with status "(proved)". It includes proof steps: (skosimp*) and (grind).
- PVS PROOF MATE:** Shows hints: (skosimp*) and (skeep), and a Sketchpad.

Prover Console:

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' X

```
Starting prover session for always_positive

always_positive :

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

>> s█
skeep
```

Proof commands are entered in the Prover Console

Use TAB to autocomplete proof commands

INTEGRATED HELP

pvs helloworld.pvs pvs Proving 'always_positive' X

Starting prover session for always_positive

always_positive :

{1} ┌── FORALL (x: real): abs(x) >= 0

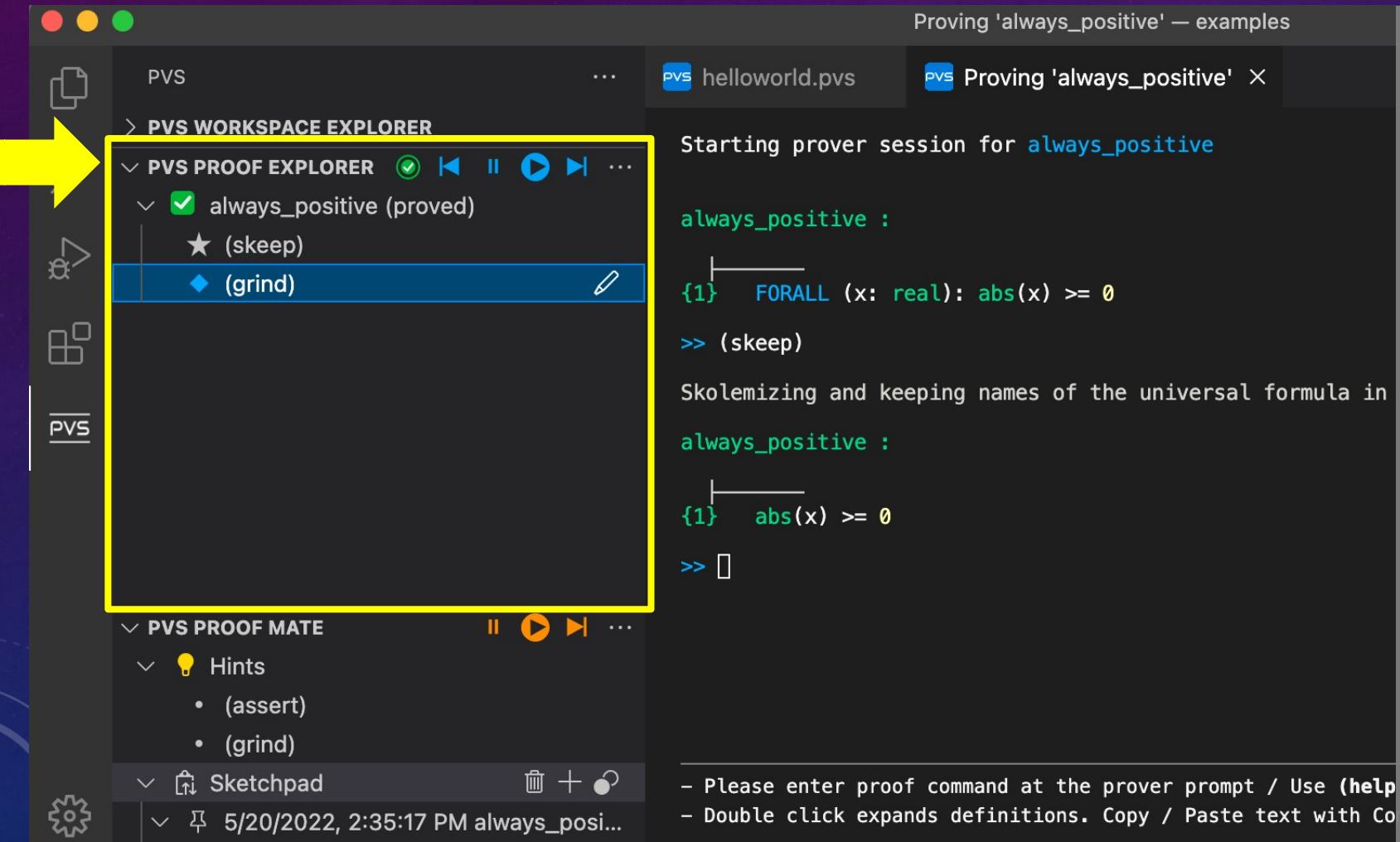
>> s█

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



Proof Explorer shows the proof tree

PROOF EXPLORER / CONTEXT MENU

Proving 'always_positive' — examples

PVS helloworld.pvs Proving 'always_positive' X

Starting prover session for `always_positive`

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

Simplifying 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 commands
Double click expands definitions. Copy / Paste text with Command-C / Command-V

The screenshot shows the PVS IDE interface with the Proof Explorer expanded. A context menu is open over a proof step. The menu items are:

- Run subtree
- Fast forward here** (highlighted with a yellow arrow and selection bar)
- Rewind here
- Jump here
- Show sequent
- Add new proof command
- Copy
- Cut
- Paste
- Copy subtree
- Cut subtree

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

Edit and copy/paste operations are restricted

PROOF MATE

The screenshot shows the Proof Mate interface within a PVS workspace. The left sidebar has sections for PVS WORKSPACE EXPLORER, PVS PROOF EXPLORER, and PVS PROOF MATE. The PVS PROOF MATE section is highlighted with a yellow arrow and a border, containing options like Hints (assert, grind), Sketchpad, and a log entry for a proof session on 'always_positive' at 5/20/2022, 2:35:17 PM.

Proving 'always_positive' — examples

PVS helloworld.pvs PVS Proving 'always_positive' ×

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 commands

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!

The screenshot shows the PVS IDE interface. On the left is the PVS WORKSPACE EXPLORER, which lists a file named "helloworld.pvs" and a proof session titled "Proving 'always_positive'". The PVS PROOF EXPLORER on the right shows a tree of proof steps for the theorem "always_positive". The steps include a FORALL quantifier over a real variable x, followed by a (skeep) step and a (grind) step. A yellow arrow points to the "Q.E.D." comment at the end of the proof tree. The status bar at the bottom indicates that the proof was completed successfully and can be closed.

```
Proving 'always_positive' — examples
PVS
PVS WORKSPACE EXPLORER
PVS PROOF EXPLORER
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
        >> (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
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 13:31:5
  prove | status-proofchain | show-prooflite
  always_positive: THEOREM
    FORALL (x: real): abs(x) >= 0
  % Example markdown expressions that can be used in the documentation
  % $\sum_{i=1}^n X_i$  
% $k_{n+1}$  
% $n^2$  
% $k_n^2$  
% $\frac{4z^3}{16}$  
% $\frac{n!}{k!(n-k)!}$  
% $\binom{n}{k}$  
% $\frac{\frac{x}{1}}{x-y}$  
% $^3\sqrt{7}$  
% $\sqrt{k}$  
% $\sqrt[n]{k}$  
% $\sum_{i=1}^{10} t_i$  
% $\int_0^\infty \mathrm{e}^{-x}, \mathrm{d}x$  
% $\hat{a}$  
% $\bar{a}$  
% $\dot{a}$  
% $\ddot{a}$  
% $\overrightarrow{AB}$  
open(sqrt_newton(n)) | view-as-markdown
% ![[sqrt_newton(n)].(sqrt_newton_n.png)]
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
helloworld: THEORY
BEGIN
% utility function, computes the absolute value of a number
abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
FORALL (x: real): abs(x) >= 0
% Example markdown expressions that can be used in the documentation
%  $\sum_{i=1}^n X_i$ 
%  $k_{n+1}$ 
%  $n^2$ 
%  $k_n^2$ 
%  $\frac{4z^3}{16}$ 
%  $\frac{n!}{k!(n-k)!}$ 
%  $\binom{n}{k}$ 
%  $\frac{1}{x-y}$ 
%  $3/7$ 
%  $\sqrt{k}$ 
%  $\sqrt[n]{k}$ 
%  $\sum_{i=1}^{10} t_i$ 
%  $\int_0^\infty e^{-x} dx$ 
%  $\hat{a}$ 
%  $\bar{a}$ 
%  $\dot{a}$ 
%  $\ddot{a}$ 
%  $\overrightarrow{AB}$ 
open(sqrt_newton(n)) | view-as-markdown
! [sqrt_newton(n)](sqrt_newton_n.png)

```

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

```

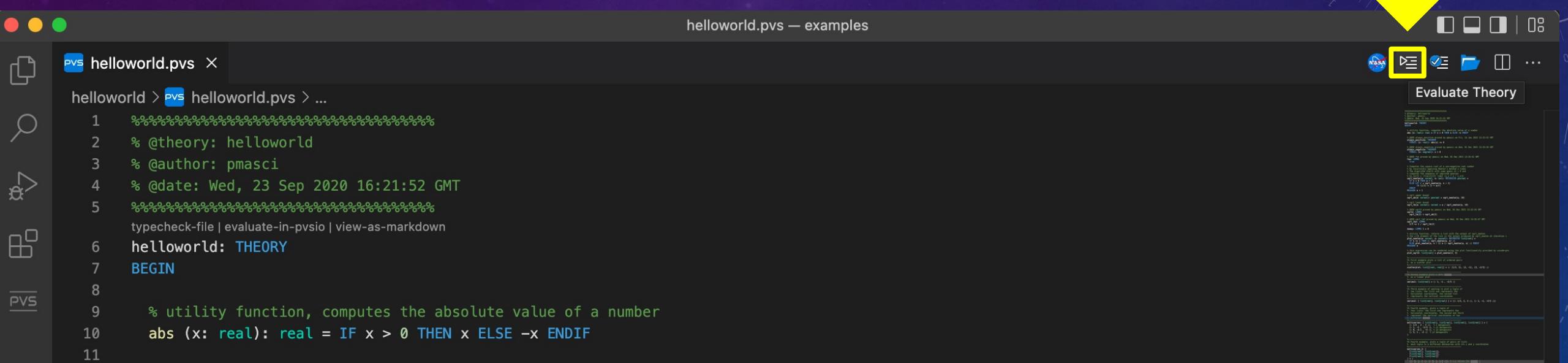
helloworld: THEORY
BEGIN
% utility function, computes the absolute value of a number
abs (x: real): real = IF x > 0 THEN x ELSE -x ENDIF
% @QED always_positive proved by pmasci on Fri, 14 Jan 2022 13:31:52 GMT
always_positive: THEOREM
FORALL (x: real): abs(x) >= 0

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

```

EVALUATING PVS EXPRESSIONS

Click the 'play' icon in the editor toolbar



helloworld.pvs — examples

PVS 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-pvsi | 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
```

Evaluate Theory

EVALUATING PVS EXPRESSIONS

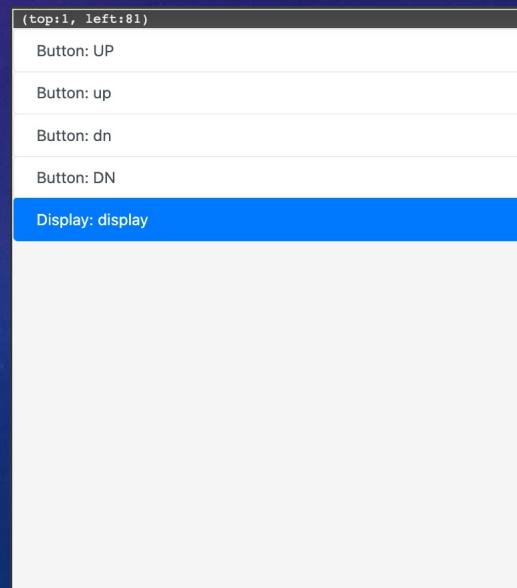
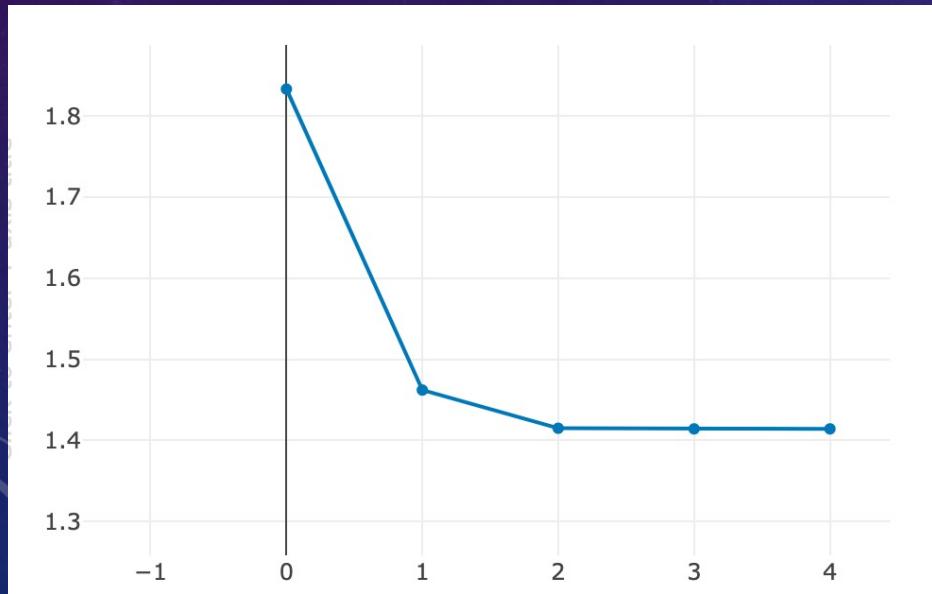
```
pvs helloworld.pvs ×  
helloworld > pvs helloworld.pvs > ...  
1 %@theory: helloworld  
2 %@author: pmasci  
3 %@date: Wed, 23 Sep 2020 16:21:52 GMT  
4  
5 typecheck-file | evaluate-in-pvsi | view-as-markdown  
6 helloworld: THEORY  
7 BEGIN  
8  
9 % utility function, computes the absolute value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

Enter ground expressions at the evaluator prompt

```
... pvs Evaluating helloworld × ...  
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
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: nnreal, 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_ne
38  plot_newton(a: nnreal, 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_sqrt2: list[real] = plot_newton(2, 5)
45
```

Define a function, e.g., 'sqrt_newton'

PLOTTING FUNCTIONS

```
helloworld > pvs helloworld.pvs > helloworld
    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      %  $x_{n+1} = \frac{1}{2} (x_n + a / x_n)$ 
29      sqrt_newton(a: nnreal, 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: nnreal, 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

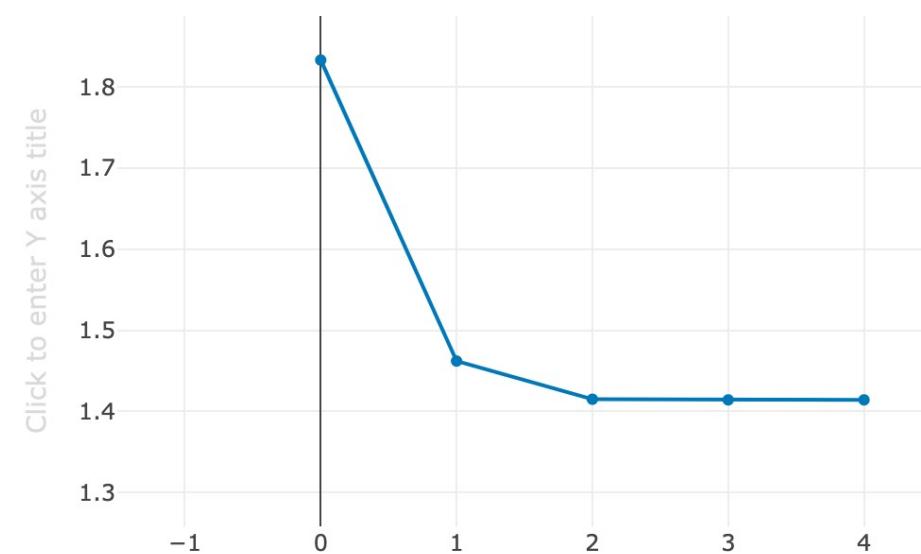
```
helloworld > pvs helloworld.pvs > helloworld
22      true
23
24      % Computes the square root of a non-negative real number
25      % by recursively applying Newton's method n times
26      % The algorithm starts with some guess x1 > 0 and
27      % computes the sequence of improved guesses
28      % $x_{n+1} = \frac{1}{2} \cdot (x_n + a / x_n)$
29      sqrt_newton(a: nnreal, 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          % This is the implementation of the function produced by the
37          % plot_newton(a, n) command. It is recursive and produces a
38          % list of real numbers representing the sequence of improved
39          % guesses.
40          plot_newton(a, n) 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)
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
pvsio-web-examples > dataEntry > AlarisGP > alarisGP.pvs > alarisGP

33     state: TYPE = [# display: alaris_real, timer: alaris_timer, step: alaris_step, power_led: ledType #]
34
35     init(x: alaris_real): state = (# display := x, timer := max_timer, step := small_step, power_led := OFF #)
36     %
37
38
39
40     %% utility functions
41     trim(x: real): alaris_real = IF x > max THEN max ELSEIF 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         ELSEIF val >= 100 AND val < 1000 THEN trim( floor((val) + delta) )
50         ELSE trim( (floor(val/10) + delta) * 10 ) ENDIF
51

```

PVS: Web Prototyping Toolkit

Picture Run

(top:1, left:81)

- Button: UP
- Button: up
- Button: dn
- Button: DN
- Display: display

Simulator Builder Settings

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/pvsio/web/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)