## **PythonTutor.com - unsupported features and FAQ**

If you are an **instructor** or instructional content creator, [**read this instructor FAQ**](https://docs.google.com/document/d/1quk8gdvgzaYrZaOSiVzCia8tCXBFmuTk0awMw_-OTuE/edit?usp=sharing).

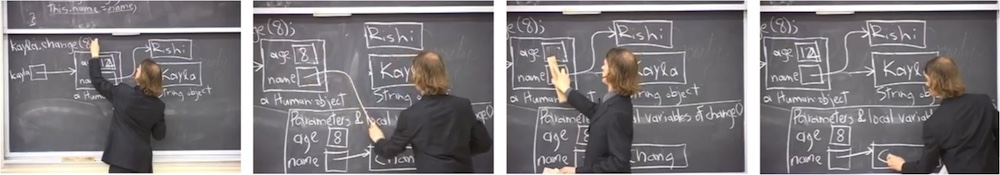
## **Reporting bugs**

The issue you’re encountering is likely listed in this document. If you're sure it's not here, use the "Generate permanent link" button to make a URL of your code. Describe the expected behavior when running that code on your computer and how it differs from Python Tutor, then [**fill out this Google Form**](https://docs.google.com/forms/d/e/1FAIpQLScGHzS7yDtHt9vmkRUp-FHCLK3NRg0hGolO7WFrqKZDyqL3Rg/viewform?usp=sf_link) to report your bug or security issue.

* This form is not for requests or questions about desired features; it is only for reproducible bug reports and private disclosures of security-related issues.
* If you don't get a reply, assume your issue will *not* be addressed. Please do not submit duplicate issues in the form.
* There is no support for Python Tutor visualizations that are embedded in other people’s websites. Contact those site owners for help on how to use their sites.

## **Unsupported features**

### **Read this first**

[Python Tutor](https://pythontutor.com/) is designed to imitate what an instructor in an introductory programming class draws on the blackboard:

It's meant to illustrate small pieces of self-contained code that runs for not too many steps. After all, an instructor can't write hundreds of lines of code, draw hundreds of data structures and pointers, or walk through hundreds of execution steps on the board! Also, code in introductory classes usually doesn't access external libraries. ***If your code can't fit on a blackboard or presentation slide, it's probably too long to visualize effectively in Python Tutor.*** This tool is not meant as a professional-level debugger.

*A more formal theory of action for how this tool works*: Python Tutor helps learners understand and debug code written for programming courses by replacing the two most common run-time inspection techniques – print statements and debuggers – with a zero-setup user interface that visualizes all run-time state at all executed lines.

Due to this ultra-focused design, the following features are not supported on purpose:

* Code that is too large in size
  + [shorten your code](https://stackoverflow.com/help/minimal-reproducible-example) to what fits on a blackboard or presentation slide
  + Python Tutor is ***not*** for debugging arbitrary code that you paste into it; you'll need to shorten your code to isolate what you want to debug
* Code that runs for too many steps (e.g., > 100) or for a long time (e.g., > 10 sec)
  + [shorten your code](https://stackoverflow.com/help/minimal-reproducible-example) to isolate exactly what operations you want to visualize
  + e.g., make your numbers/strings smaller, arrays/lists shorter, your data structures contain fewer items, and your loops/functions run fewer times
  + for Python, set breakpoints using special #break comments ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%0A%23%20when%20the%20string%20%23break%20is%20the%20last%20thing%20on%20a%20line,%20then%20only%0A%23%20trace%20the%20execution%20of%20those%20lines.%20this%20can%20make%20it%20possible%20to%0A%23%20trace%20longer-running%20programs%20by%20stopping%20only%20at%20selected%20lines%0A%0A%23%20normally%20execution%20would%20halt%20after%201000%20steps,%20so%20you'd%20never%0A%23%20get%20to%20print%20everything%20...%20but%20if%20you%20put%20'%23break'%20on%20line%2012%0A%23%20then%20it%20will%20stop%20and%20print%20out%20all%20the%20desired%20numbers%0Afor%20i%20in%20range%2810000%29%3A%0A%20%20%20%20if%20i%20%3E%200%20and%20i%20%25%201000%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20print%28i%29%20%23break&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false))
* Code that defines too many variables or objects
  + [shorten your code](https://stackoverflow.com/help/minimal-reproducible-example) to isolate what variables you want to visualize
  + remove unnecessary variables and objects from your code
  + for Python, use #pythontutor\_hide to selectively hide objects ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%0A%23%20use%20'%23pythontutor_hide'%20to%20pass%20in%20a%20glob%20expression%0A%23%20%28https%3A//docs.python.org/3/library/glob.html%29%20for%20what%0A%23%20variable%20names%20to%20hide.%20in%20this%20example,%20'*Type'%20means%20hide%0A%23%20all%20variables%20that%20end%20with%20'Type'.%20'y*'%20means%20anything%20starting%20with%20'y'%0A%0A%23%20use%20'%23pythontutor_hide_type'%20to%20pass%20in%20names%20of%20types%20of%20objects%0A%23%20to%20hide.%20in%20this%20example,%20we%20hide%20all%20classes,%20functions,%20and%20tuples%0A%0A%23%20if%20you%20remove%20these%20comments%20below,%20you%20will%20see%20how%20the%20visualization%0A%23%20is%20overly%20complicated.%20using%20these%20can%20drastically%20reduce%20the%20number%0A%23%20of%20visualized%20objects%20to%20a%20manageable%20amount.%0A%0A%23pythontutor_hide%3A%20*Type,%20y*%0A%23pythontutor_hide_type%3A%20class,%20function,%20tuple%0A%0A%0Afrom%20types%20import%20*%20%23%20lots%20of%20stuff%20in%20here%20should%20be%20hidden%0A%0Adef%20func1%28%29%3A%0A%20%20%20%20pass%0Adef%20func2%28%29%3A%0A%20%20%20%20pass%0A%0Ax%20%3D%20%281,2,3%29%0A%23%20all%20the%20y's%20should%20be%20hidden%0Ay%20%3D%20list%28x%29%0Ay2%20%3D%20y%0Ay3%20%3D%20y%0Az%20%3D%20y%20%23%20'z'%20should%20be%20the%20only%20object%20shown%20in%20the%20visualization%20%3A%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false))
  + also use “Move and hide objects” option at bottom-left of visualizer to hide
* Advanced language features or subtleties that only experts need to know about
* Importing most external libraries (it’s meant for learning basic coding concepts)
* Visualizing custom data structures from libraries (it supports only built-in types)
* Interfacing with files, databases, networking, or other external resources
* Anything involving GUI programming or manipulating GUI/webpage components
* Multi-threaded / concurrent / asynchronous code (only supports single-threaded)

### **Other general unsupported features**

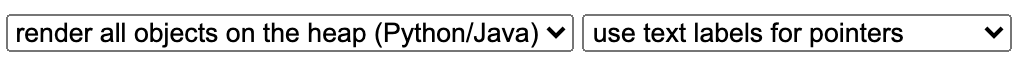
* Command-line arguments (argv[]) not supported; use hard-coded strings instead
* Reading data from external files is not supported (workaround: use strings to emulate files. StringIO examples for [Python3](http://goo.gl/uNvBGl) and [Python2](http://goo.gl/Q9xQ4p))
* You cannot step *within* a line of code to show how subexpressions get evaluated within that line; the best workaround is to manually split complex expressions into multiple lines and assign to temporary variables on each line ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0Aw%20%3D%205%0Ax%20%3D%2010%0Ay%20%3D%2020%0Az%20%3D%2030%0A%0A%23%20bad%3A%20executes%20all%20at%20once%20since%20Python%20Tutor%20steps%20over%20the%20entire%20line%0Aresult%20%3D%20w%20-%20x%20*%20%28y%20%2B%20z%29%0A%0A%23%20good%3A%20break%20up%20an%20expression%20into%20its%20subexpressions%20to%20show%20individual%20steps%0At1%20%3D%20y%20%2B%20z%0At2%20%3D%20x%20*%20t1%0Aresult2%20%3D%20w%20-%20t2&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)).
* Printing to stderr probably won’t work; use print statements to print to stdout
* Some Unicode characters may not display if your browser doesn’t have those fonts or if you’re trying to print *unprintable characters* like binary data to terminal
* This tool uses slightly older versions of languages (e.g., Python 3.6) for greater stability and because instructional materials often rely on older versions. Upgrading to the newest versions can confuse beginners who are learning from instructional materials since the compiler/interpreter messages do not match their materials.

### 

### **Python known limitations**

* (scroll up to see general limitations and workarounds on the previous pages)
* Python 3.6 is well-tested; Python 3.11 is new so *newer features may not work*!
  + 3.11’s [improved run-time error messages](https://peps.python.org/pep-0657/) are *not* shown due to limitations in debugger, but improved syntax error messages are shown
* For strings and numbers, you can't rely on id() or is behaving identically to CPython on your computer ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0Ax%20%3D%201000%0Ay%20%3D%201000%0A%23%20True%20here%3B%20may%20be%20False%20in%20CPython%20REPL,%20but%20True%20in%20scripts%0A%23%20%28behavior%20also%20differs%20if%20x%20and%20y%20are%20*small*%20numbers%29%0A%23%20lesson%3A%20do%20*not*%20rely%20on%20is%28%29%20or%20id%28%29%20behaving%20identically%20across%20settings%0Aprint%28x%20is%20y%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)); when teaching beginners, *never rely* on these since they’re implementation-specific and differ between REPL and scripts
* Don’t put code with side effects (e.g., print) or mutations in \_\_str\_\_ or \_\_repr\_\_, since Python Tutor will call those to get object values (just like a debugger does), which makes them run more times than what your code specifies ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0Aclass%20Foo%28object%29%3A%0A%20%20%20%20def%20__init__%28self,%20name%29%3A%0A%20%20%20%20%20%20%20%20self.name%20%3D%20name%0A%20%20%20%20%20%20%20%20self.counter%20%3D%200%0A%0A%20%20%20%20%23%20do%20*not*%20put%20side%20effects%20or%20mutations%20in%20__str__%20or%20__repr__%0A%20%20%20%20%23%20since%20Python%20Tutor%20will%20call%20these%20functions%20to%20display%0A%20%20%20%20%23%20visualization%20elements,%20so%20they%20will%20be%20called%20more%20times%0A%20%20%20%20%23%20than%20your%20code%20actually%20calls%20them%0A%20%20%20%20def%20__str__%28self%29%3A%0A%20%20%20%20%20%20%20%20print%28self.name,%20self.counter%29%20%20%23%20side-effect!%0A%20%20%20%20%20%20%20%20self.counter%20%2B%3D%201%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%23%20mutation!%0A%20%20%20%20%20%20%20%20return%20f'%5Bname%3A%20%7Bself.name%7D,%20counter%3A%20%7Bself.counter%7D%5D'%0A%0Af%20%3D%20Foo%28'Bobby'%29%0Aprint%28f%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false))
* #ambiguousParentFrame - parent frames and closures may sometimes be shown inaccurately if multiple ones “look identical” from the current frame’s POV; CPython does not expose this data to tools, so Python Tutor must use heuristics:
  + Walkthrough of #ambiguousParentFrame issue: [Part1](https://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20Python%20Tutor%20must%20use%20heuristics%20to%20track%20parent%20frames,%20and%20sometimes%20it%20gets%20it%20wrong%0A%0A%23%20Example%201%20of%204%0A%0Adef%20makefunc%28x%29%3A%0A%20%20%20%20def%20g%28%29%3A%0A%20%20%20%20%20%20%20%20print%28%22I%20am%20using%20x%3A%22,%20x%29%0A%20%20%20%20return%20g%0A%0A%23%20if%20you%20create%20more%20than%20one%20parent%20with%20the%20same%20local%20variable%0A%23%20values,%20then%20there's%20no%20way%20for%20Python%20Tutor%20to%20tell%20them%20apart%0Ag1%20%3D%20makefunc%283%29%0Ag2%20%3D%20makefunc%283%29%0Ag2%28%29%20%23%20parent%20should%20be%20f2,%20not%20f1%20%28but%20both%20look%20identical%29%0A%23%20when%20g%20accesses%20x%20to%20print%20it%20out,%20it%20will%20print%203,%20but%20both%20f1%20and%20f2%0A%23%20could've%20contributed%20that%20x%3D3%20to%20g,%20so%20there's%20no%20way%20for%20Python%20Tutor%0A%23%20to%20tell%20that%20f2%20is%20indeed%20the%20real%20parent.%20the%20CPython%20interpreter%0A%23%20simply%20doesn't%20track%20that%20information%20%3A/&cumulative=false&curInstr=14&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false), [Part2](https://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20Python%20Tutor%20must%20use%20heuristics%20to%20track%20parent%20frames,%20and%20sometimes%20it%20gets%20it%20wrong%0A%0A%23%20Example%202%20of%204%0A%0Adef%20makefunc%28x%29%3A%0A%20%20%20%20def%20g%28%29%3A%0A%20%20%20%20%20%20%20%20print%28%22I%20didn't%20use%20x!%22%29%20%23%20%3C--%20code%20changed%20to%20*not*%20use%20x%20now%0A%20%20%20%20return%20g%0A%0A%23%20if%20you%20create%20more%20than%20one%20parent%20with%20the%20same%20local%20variable%0A%23%20values,%20then%20there's%20no%20way%20for%20Python%20Tutor%20to%20tell%20them%20apart%0A%23%0A%23%20...%20so%20one%20workaround%20is%20to%20create%20frames%20with%20different%20values%3A%0Ag1%20%3D%20makefunc%283%29%0Ag2%20%3D%20makefunc%284%29%0Ag2%28%29%20%23%20parent%20should%20be%20f2,%20not%20f1%0A%23%20...%20but%20Python%20Tutor%20*still*%20can't%20tell%20the%20difference%20here,%20though,%0A%23%20for%20a%20subtle%20reason%3A%20g%20never%20actually%20uses%20x%20from%20its%20parent%20frame,%0A%23%20so%20the%20CPython%20interpreter%20kinda%20'optimizes%20out'%20its%20value%20when%0A%23%20calling%20g,%20which%20makes%20it%20like%20x%20never%20existed.%20Technically%20there's%0A%23%20no%20way%20when%20you're%20in%20g%20to%20tell%20who%20your%20parent%20is,%20since%20you're%0A%23%20not%20accessing%20the%20value%20of%20x%20from%20it,%20so%20it%20doesn't%20matter%20if%20it's%0A%23%20f1%20or%20f2,%20with%20x%20as%203%20or%204,%20respectively.%20You%20never%20use%20it!&cumulative=false&curInstr=14&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false), [Part3](https://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20Python%20Tutor%20must%20use%20heuristics%20to%20track%20parent%20frames,%20and%20sometimes%20it%20gets%20it%20wrong%0A%0A%23%20Example%203%20of%204%0A%0Adef%20makefunc%28x%29%3A%0A%20%20%20%20def%20g%28%29%3A%0A%20%20%20%20%20%20%20%20print%28%22I%20am%20using%20x%3A%22,%20x%29%20%23%20%3C--%20ok%20we're%20using%20x%20again%0A%20%20%20%20return%20g%0A%0A%23%20if%20you%20create%20more%20than%20one%20parent%20with%20the%20same%20local%20variable%0A%23%20values,%20then%20there's%20no%20way%20for%20Python%20Tutor%20to%20tell%20them%20apart%0A%23%20...%20one%20workaround%20is%20to%20create%20frames%20with%20different%20values%3A%0Ag1%20%3D%20makefunc%283%29%0Ag2%20%3D%20makefunc%284%29%0Ag2%28%29%20%23%20ok,%20now%20Python%20Tutor%20can%20definitively%20tell%20that%20the%20parent%20is%20f2,%0A%23%20since%20x%20is%20being%20used%20by%20g,%20so%20CPython%20can't%20'optimize%20out'%20the%20value%20of%20x.%0A%23%20It%20now%20matters%20whether%20x%3D3%20%28from%20f1%29%20or%20x%3D4%20%28from%20f2%29,%20and%20here%20it's%0A%23%20definitely%20x%3D4,%20so%20Python%20Tutor%20determines%20that%20f2%20is%20the%20parent.&cumulative=false&curInstr=14&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false), [Part4](https://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20Python%20Tutor%20must%20use%20heuristics%20to%20track%20parent%20frames,%20and%20sometimes%20it%20gets%20it%20wrong%0A%0A%23%20Example%204%20of%204%0A%0Adef%20makefunc%28x%29%3A%0A%20%20%20%20def%20g%28%29%3A%0A%20%20%20%20%20%20%20%20x%20%3D%2010%20%23%20%3C--%20a%20local%20x%20'shadows'%20the%20x%20from%20my%20parent%0A%20%20%20%20%20%20%20%20print%28%22I%20am%20now%20using%20a%20local%20x%3A%22,%20x%29%0A%20%20%20%20return%20g%0A%0A%23%20if%20you%20create%20more%20than%20one%20parent%20with%20the%20same%20local%20variable%0A%23%20values,%20then%20there's%20no%20way%20for%20Python%20Tutor%20to%20tell%20them%20apart%0A%23%20...%20one%20workaround%20is%20to%20create%20frames%20with%20different%20values%3A%0Ag1%20%3D%20makefunc%283%29%0Ag2%20%3D%20makefunc%284%29%0Ag2%28%29%20%23%20now%20Python%20Tutor%20gets%20confused%20again%20because%20we%20declared%20a%20local%20x%0A%23%20which%20shadows%20the%20x%20from%20its%20parent%20frame.%20Thus,%20we're%20back%20in%20the%20same%0A%23%20situation%20as%20Example%202,%20when%20g%20didn't%20use%20x%20from%20its%20parent%20frame.%20Since%0A%23%20g%20uses%20its%20own%20local%20x,%20it's%20no%20longer%20using%20x%20from%20makefunc,%20so%20again%0A%23%20CPython%20doesn't%20record%20whether%20its%20parent%20is%20f1%20%28x%3D3%29%20or%20f2%20%28x%3D4%29,%20because%0A%23%20it%20doesn't%20really%20matter%20which%20one%20it%20is.%20g%20is%20using%20its%20own%20local%20x%3D10%0A%23%20regardless.%20Hence,%20Python%20Tutor%20can't%20tell%20the%20difference.%20The%20end!%0A%23%0A%23%20Summary%3A%20if%20you%20want%20Python%20Tutor%20to%20do%20the%20most%20accurate%20tracing,%0A%23%20create%20parent%20frames%20that%20have%20different%20local%20variable%20values,%0A%23%20and%20access%20them%20from%20inside%20the%20current%20frame%20*without%20shadowing%20them*.&cumulative=false&curInstr=15&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)
    - additional examples: [ex1](https://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20Python%20Tutor%20must%20use%20heuristics%20to%20track%20closures%20and%20parent%20frames,%20and%20sometimes%20it%20gets%20it%20wrong%0Adef%20makeadder%28n%29%3A%0A%20%20%20%20print%28n%29%0A%20%20%20%20def%20adder%28k%29%3A%0A%20%20%20%20%20%20%20%20print%28n,%20k%29%0A%20%20%20%20%20%20%20%20return%20n%20%2B%20k%0A%20%20%20%20return%20adder%0A%0Aa%20%3D%20makeadder%282%29%286%29%0Ab%20%3D%20makeadder%282%29%0A%23%20when%20b%28%29%20is%20called%20in%20Steps%2019-22,%20its%20parent%20should%20be%20f3%20and%20*not*%20f1,%20but%0A%23%20Python%20Tutor%20can't%20tell%20the%20difference%20because%20f1%20and%20f3%20have%20identical-looking%20stack%20values%0A%23%20%28change%20line%2011%20to%20makeadder%283%29%20and%20it%20will%20work%20since%20the%20two%20makeadders%20will%20have%20different%20stack%20values%29%0Ac%20%3D%20b%286%29&cumulative=false&curInstr=18&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false), [ex2](https://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20Python%20Tutor%20must%20use%20heuristics%20to%20track%20closures%20and%20parent%20frames,%20and%20sometimes%20it%20gets%20it%20wrong%0A%23%20example%20adapted%20from%20http%3A//albertwu.org/cs61a/review/hof/basic.html%0Adef%20boom%28%29%3A%0A%20%20%20%20%23%20when%20bam%28%29%20is%20called%20in%20Steps%2023-25,%20its%20parent%20frame%20should%20be%20f3,%20*not*%20f1,%20but%0A%20%20%20%20%23%20Python%20Tutor%20can't%20tell%20the%20difference%20so%20it%20guesses%20and%20warns%20with%20%23ambiguousParentFrame%0A%20%20%20%20def%20bam%28%29%3A%0A%20%20%20%20%20%20%20%20print%28%22BAM!%22%29%0A%20%20%20%20return%20bam%0A%0Aboom%28%29%20%20%20%20%20%23%20creates%20frame%20f1%20but%20never%20assigned%20to%20anything%0Aboom%28%29%20%20%20%20%20%23%20creates%20f2%20but%20never%20assigned%0Aa%20%3D%20boom%28%29%20%23%20creates%20f3,%20assigned%20to%20a%0Aboom%28%29%20%20%20%20%20%23%20creates%20f4,%20but%20never%20assigned%0Aa%28%29&cumulative=false&curInstr=22&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false), [ex3](https://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20Python%20Tutor%20must%20use%20heuristics%20to%20track%20closures%20and%20parent%20frames,%20and%20sometimes%20it%20gets%20it%20wrong%0Adef%20f%28%29%3A%0A%20%20%20%20def%20g%28%29%3A%0A%20%20%20%20%20%20%20%20pass%0A%20%20%20%20return%20g%0A%0Af%28%29%28%29%20%23%20parent%20is%20f1%20since%20that's%20the%20only%20one%0Af%28%29%28%29%20%23%20Python%20Tutor%20can't%20tell%20whether%20parent%20is%20f1%20or%20f3&cumulative=false&curInstr=15&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)
  + parent frame can get “lost” when assigned to an object field: [example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20Python%20Tutor%20must%20use%20heuristics%20to%20track%20closures%20and%20parent%20frames,%20and%20sometimes%20it%20gets%20it%20wrong%0Aclass%20Foo%3A%0A%20%20%20%20def%20__init__%28self,%20a%29%3A%0A%20%20%20%20%20%20%20%20def%20inner%28b%29%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20a%2Bb%0A%20%20%20%20%20%20%20%20%23%20parent%20frame%20'f1'%20gets%20lost%20when%20inner%20is%20assigned%20to%20a%20field%0A%20%20%20%20%20%20%20%20self.foo%20%3D%20inner%0A%20%20%20%20%20%20%20%20local_lambda%20%3D%20lambda%20b%3A%20a%2Bb%0A%20%20%20%20%20%20%20%20%23%20parent%20frame%20'f1'%20gets%20lost%20when%20local_lambda%20is%20assigned%20to%20a%20field%0A%20%20%20%20%20%20%20%20self.bar%20%3D%20local_lambda%0A%0Af%20%3D%20Foo%2810%29%0Af.bar%2820%29%20%23%20parent%20frame%20'f1'%20is%20shown%20fine%20here&cumulative=true&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)
* can’t differentiate between multiple lambdas defined on one line: [example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20Python%20Tutor%20must%20use%20heuristics%20to%20track%20closures%20and%20parent%20frames,%20and%20sometimes%20it%20gets%20it%20wrong%0A%0Adef%20f%28a,%20b,%20c%29%3A%0A%20%20%20%20a_res%20%3D%20a%2810%29%0A%20%20%20%20b_res%20%3D%20b%2820%29%0A%20%20%20%20c_res%20%3D%20c%2830%29%0A%0A%23%20it%20can't%20differentiate%20between%20the%20first%20and%20third%20lambdas%20since%20they%20look%20identical%3A%0Aw%20%3D%20f%28lambda%20x%3Ax*x,%20lambda%20y%3Ay*y,%20lambda%20x%3Ax*x%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)
* \_\_del\_\_ is not guaranteed to run at the same times that it does in CPython since Python Tutor holds onto additional references to objects to create accurate traces ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0Aclass%20Example%3A%0A%20%20%20%20def%20__init__%28self%29%3A%0A%20%20%20%20%20%20%20%20print%28%22Example%20instance.%22%29%0A%20%20%20%20def%20__del__%28self%29%3A%0A%20%20%20%20%20%20%20%20print%28%22Finalizer%20called,%20Example%20deleted.%22%29%0A%0Aobj%20%3D%20Example%28%29%0Aprint%28'before%20del%20...'%29%0Adel%20obj%0Aprint%28'...%20after%20del'%29%0A%23%20__del__%20is%20*not*%20guaranteed%20to%20run%20after%20a%20'del'%20call,%20even%20if%20no%0A%23%20other%20references%20to%20it%20exist.%20see%20https%3A//stackoverflow.com/a/2452895&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)); in general, [you can’t rely on \_\_del\_\_ running](https://stackoverflow.com/a/2452895) at any specific time.
* Random number generators always start at the same fixed seed for reproducible results, but that makes it harder to test code that relies on randomness ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23%20random%20numbers%20always%20start%20at%20same%20fixed%20seed,%20so%20it%20will%20produce%0A%23%20the%20same%20numbers%20even%20across%20different%20runs%0Aimport%20random%0Ax%20%3D%20random.randint%281,%2010%29%0Ay%20%3D%20input%28%29%0Az%20%3D%20random.randint%281,%2010%29%0Aw%20%3D%20random.randint%281,%2010%29%0Aprint%28x,%20y,%20z,%20w%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%22324234%22%5D&textReferences=false))
* Python Tutor does not track or visualize events that occur at the very end of execution, such as GeneratorExit exceptions or garbage collector runs ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0Adef%20my_generator%28%29%3A%0A%20%20%20%20try%3A%0A%20%20%20%20%20%20%20%20while%20True%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20print%28'print%20stuff'%29%0A%20%20%20%20%20%20%20%20%20%20%20%20token%20%3D%20%28yield%29%0A%20%20%20%20%23%20this%20is%20raised%20at%20the%20very%20end%20of%20execution%20after%0A%20%20%20%20%23%20Python%20Tutor%20stops%20tracing,%20so%20this%20will%20not%20be%20printed%0A%20%20%20%20%23%20%28it%20will%20print%20in%20CPython,%20though%29%0A%20%20%20%20except%20GeneratorExit%3A%0A%20%20%20%20%20%20%20%20print%28%22Done%20with%20printing!%22%29%0A%0Apt%20%3D%20my_generator%28%29%0Apt.__next__%28%29%0A&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false))
* will not track line numbers when running code inside of eval or exec ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0Aprint%28'hello'%29%0Aprint%28'this%20is%20line%203'%29%0A%23%20Python%20Tutor%20will%20not%20accurately%20track%20line%20numbers%20when%20running%0A%23%20code%20in%20exec%28%29%20or%20eval%28%29,%20so%20the%20line%20numbers%20below%20will%20be%20wrong%3A%0Aexec%28'x%20%3D%201%5Cny%20%3D%202%5Cnz%20%3D%20x%20%2B%20y%5Cnprint%28%22not%20tracking%20lines%20properly!%22%29'%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false))
* If your code calls lots of import statements within function calls, the visualizer may give a server error ([example](http://pythontutor.com/visualize.html#code=%23%20%5Bofficial%20Python%20Tutor%20example%5D%0Adef%20foo%28%29%3A%0A%20%20%20%20%23%20do%20*not*%20do%20this,%20since%20it%20will%20call%20'import'%20many%20times%0A%20%20%20%20%23%20and%20its%20value%20does%20not%20get%20cached,%20unlike%20in%20CPython%0A%20%20%20%20%23%20%28instead,%20move%20'import%20re'%20to%20the%20top%20as%20a%20global%20import%29%0A%20%20%20%20import%20re%0A%20%20%20%20%0Afor%20i%20in%20range%28100%29%3A%0A%20%20%20%20foo%28%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)); call import statements at top level if possible
* In general you can’t import any modules, but these *may* work: \_\_future\_\_, abc, array, bisect, calendar, cmath, collections, copy, datetime, decimal, doctest, fractions, functools, hashlib, heapq, io, itertools, json, locale, math, operator, pickle, pprint, random, re, string, types, typing, unittest

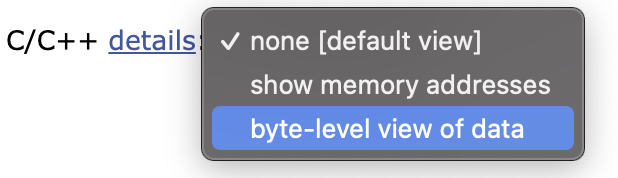
FAQ: I thought all objects in Python are (conceptually) on the heap, so why does Python Tutor render primitive values (e.g., numbers, strings, booleans) directly inside of stack frames?

This was a design decision purposely made to **keep the display less cluttered**; if this tool were truly faithful to Python's semantics, that would result in far too many arrows (pointers) being drawn. However, note that since primitives are immutable and thus behave identically regardless of aliasing, it doesn't matter whether they're rendered in the stack or heap. That said, below the code editor there is a menu option to "render all objects on the heap.” To avoid too many arrows being drawn, you can also toggle the "draw pointers as arrows" option to "use text labels for pointers":

### 

### **C and C++ known limitations**

[**Summary of C/C++ features**](https://pythontutor.com/articles/c-cpp-visualizer.html). Compiles with gcc 9.3 and runs with [Valgrind Memcheck](https://www.valgrind.org/docs/manual/mc-manual.html) plugin. Supports C17 and C++20 + GNU extensions but **works better with C**. It has limitations due to the difficulty of safely tracing C/C++. Here is what it does *not* support:

* doesn't show function return values ([remedy: assign return value to a variable](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0A%0A//%20Python%20Tutor%20doesn't%20visualize%20return%20values%20from%20functions%20...%0Aint%20foo%28a,%20b%29%20%7B%0A%20%20printf%28%22calling%20with%20%25d%20%25d%5Cn%22,%20a,%20b%29%3B%0A%20%20return%20a%20*%20b%3B%0A%7D%0A%0A//%20...%20so%20the%20workaround%20is%20to%20assign%20the%20return%20value%20to%20a%20variable%0Aint%20foo2%28a,%20b%29%20%7B%0A%20%20printf%28%22calling%20with%20%25d%20%25d%5Cn%22,%20a,%20b%29%3B%0A%20%20int%20_return%20%3D%20a%20*%20b%3B%0A%20%20return%20_return%3B%0A%7D%0A%0Aint%20main%28%29%20%7B%0A%20%20int%20x%20%3D%20foo%2810,%2020%29%3B%0A%20%20int%20y%20%3D%20foo2%2810,%2020%29%3B%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false))
* can’t take text input from the user using scanf(), fgets with stdin, cin >>, etc.
* checks for memory safety with [Memcheck](https://www.valgrind.org/docs/manual/mc-manual.html), which leads to these behaviors:
  + leaked memory is not visualized since nothing points to it anymore
  + code with [undefined behavior](https://blog.regehr.org/archives/213) (e.g., [uninitialized/unallocated values](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0A%0A//%20this%20code%20shows%20what%20question%20marks%20%28'%3F'%29%20and%20skull%20emojis%20mean%20in%20the%20visualization%0Atypedef%20struct%20%7B%0A%20%20int%20a%3B%0A%20%20int%20b%3B%0A%20%20int%20c%3B%0A%20%20int%20d%3B%0A%20%20int%20e%3B%0A%7D%20foo%3B%0A%0Aint%20main%28%29%20%7B%0A%20%20foo*%20p_foo%20%3D%20%28foo*%29malloc%28sizeof%28*p_foo%29%29%3B%20%20//%20right%20size!%20allocates%20enough%20space%20for%20the%20entire%20struct%0A%20%20//%20question%20marks%20'%3F'%20mean%20allocated%20but%20uninitialized%20memory,%20which%20is%20fine%20for%20now%0A%0A%20%20foo*%20p_foo2%20%3D%20%28foo*%29malloc%28sizeof%28p_foo2%29%29%3B%20//%20wrong%20size!%20allocates%20only%20enough%20space%20for%20a%20pointer%20to%20struct!%0A%20%20//%20skulls%20mean%20UNALLOCATED%20memory%0A%0A%20%20//%20note%20that%20the%20'%3F'%20are%20replaced%20by%20legitimate%20integer%20values%20...%0A%20%20p_foo-%3Ea%20%3D%201%3B%0A%20%20p_foo-%3Eb%20%3D%202%3B%0A%20%20p_foo-%3Ec%20%3D%203%3B%0A%20%20%0A%20%20p_foo2-%3Ea%20%3D%201%3B%0A%20%20p_foo2-%3Ec%20%3D%203%3B%20//%20error%20when%20trying%20to%20write%20to%20skull%20%28UNALLOCATED%20memory%20block%29%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false), type- and memory-unsafe operations) may run differently than on your computer
  + 💀means a pointer points to memory that is unallocated or misaligned with data boundaries. 💀locations are approximate and may not match the pointer's real address. Select 'byte-level view of data' to see details:
* pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as *all* its members. select 'show memory addresses' to see details.
* neighboring [arrays](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0A%0A//%20sometimes%20neighboring%20arrays%20on%20the%20stack%20seem%20to%20visualize%20as%20the%20same%20object%20%3A%28%0Aint%20main%28%29%20%7B%0A%20%20int%20a%5B4%5D%3B%0A%20%20int%20b%5B4%5D%3B%0A%20%20int%20c%5B4%5D%3B%0A%20%20for%20%28int%20i%3D0%3B%20i%3C4%3B%20i%2B%2B%29%7B%0A%20%20%20%20a%5Bi%5D%20%3D%20i%3B%20//%20b%20and%20c%20also%20look%20like%20they're%20updated,%20which%20is%20wrong!%0A%20%20%20%20//b%5Bi%5D%20%3D%20i*i%3B%20//%20if%20you%20uncomment%20this%20line,%20then%20only%20b%20and%20c%20seem%20aliased%0A%20%20%20%20//c%5Bi%5D%20%3D%20i*i*i%3B%20//%20if%20you%20uncomment%20this%20line,%20then%20all%203%20arrays%20look%20separate!%0A%20%20%7D%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false) or [structs](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Astruct%20person_name%7B%0A%20%20int%20first%3B%0A%7D%3B%0A%0A//%20sometimes%20neighboring%20structs%20on%20the%20stack%20seem%20to%20visualize%20as%20the%20same%20object%20%3A%28%0Aint%20main%28%29%20%7B%0A%20%20struct%20person_name%20student1,%20student2%3B%20%20%0A%20%20%0A%20%20student1.first%20%3D%2042%3B%0A%20%20//%20both%20student1.first%20and%20student2.first%20are%20set%20to%2042,%20which%20is%20wrong%0A%20%20//%20%28*maybe*%20student2%20has%20been%20OPTIMIZED%20OUT%20by%20the%20compiler%0A%20%20//%20since%20it's%20uninitialied%20and%20never%20used%3F!%3F%29%0A%20%20%0A%20%20//%20if%20you%20uncomment%20the%20line%20below,%20then%20things%20work%20fine!%0A%20%20//student2.first%20%3D%2012345%3B%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false) on the stack may incorrectly look aliased if they are uninitialized *[other tools have this limitation too: even gdb shows them as incorrectly aliased!]*
* doesn’t show some arrays like [VLAs](https://gcc.gnu.org/onlinedocs/gcc/Variable-Length.html) ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Aint%20main%28%29%20%7B%0A%20%20int%20x%20%3D%2010%3B%0A%20%20int%20arr%5Bx%5D%3B%20//%20array%20does%20not%20have%20a%20compile-time%20size,%20so%20not%20shown%20in%20visualizer%0A%20%20for%20%28int%20i%20%3D%200%3B%20i%20%3C%2010%3B%20i%2B%2B%29%20%7B%0A%20%20%20%20arr%5Bi%5D%20%3D%20i*i%3B%0A%20%20%7D%0A%20%20for%20%28int%20i%20%3D%200%3B%20i%20%3C%2010%3B%20i%2B%2B%29%20%7B%0A%20%20%20%20printf%28%22%25d%20%25d%5Cn%22,%20i,%20arr%5Bi%5D%29%3B%0A%20%20%7D%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false)), const int dimensions ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Aconst%20int%20DIM1%20%3D%203%3B%0Aconst%20int%20DIM2%20%3D%204%3B%0A%0Aint%20main%28%29%20%7B%0A%20%20//%20limitation%3A%20still%20doesn't%20show%20in%20visualization%20even%20%0A%20%20//%20though%20dimensions%20are%20declared%20as%20'const%20int'%3A%0A%20%20int%20x%5BDIM1%5D%5BDIM2%5D%3B%0A%20%20int%20i,%20j%3B%0A%20%20for%20%28i%20%3D%200%3B%20i%20%3C%20DIM1%3B%20i%2B%2B%29%0A%20%20%20%20for%28j%20%3D%200%3B%20j%20%3C%20DIM2%3B%20j%2B%2B%29%0A%20%20%20%20%20%20x%5Bi%5D%5Bj%5D%20%3D%20i%2Bj*i%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false)), char\*\* literal initializers ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0A%23include%20%3Cstdio.h%3E%0A%0Avoid%20main%28%29%20%7B%0A%20%20//%20doesn't%20display%20in%20visualizer%20but%20printf%20works%20fine%0A%20%20char**%20values%20%3D%20%28char%20*%5B%5D%29%7B%22apple%22,%20%22banana%22,%20%22cherry%22%7D%3B%0A%20%20printf%28%22%25s%5Cn%22,%20values%5B0%5D%29%3B%0A%20%20printf%28%22%25s%5Cn%22,%20values%5B1%5D%29%3B%0A%20%20printf%28%22%25s%5Cn%22,%20values%5B2%5D%29%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false)), and zero-sized arrays like “int x[0];” ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Aint%20main%28%29%20%7B%0A%20%20int%20x%5B0%5D%3B%20//%20crashes%20the%20visualizer%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false))
* [flexible array member](https://en.wikipedia.org/wiki/Flexible_array_member) - unbounded arrays as struct member variables ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0A//%20doesn't%20display%20well%20since%20we%20don't%20know%20the%20size%20of%20buf%5B%5D%0Astruct%20aaa%20%7B%0A%20%20int%20a%3B%0A%20%20char%20flag%3B%0A%20%20char%20buf%5B%5D%3B%20//%20https%3A//en.wikipedia.org/wiki/Flexible_array_member%0A%7D%3B%0A%0Aint%20main%28%29%20%7B%0A%20%20struct%20aaa%20x%20%3D%20%7B1,%20'x'%7D%3B%0A%20%20struct%20aaa*%20px%20%3D%20malloc%2850%29%3B%0A%20%20px-%3Ea%20%3D%2042%3B%0A%20%20px-%3Eflag%20%3D%20'y'%3B%0A%20%20//%20visualizer%20doesn't%20display%20this%20right%20since%20it%20has%20no%20idea%20that%20px-%3Ebuf%0A%20%20//%20is%20an%20array%20so%20it%20just%20displays%20px%20as%20an%20array%20of%20consecutive%20'struct%20aaa'%0A%20%20px-%3Ebuf%5B0%5D%20%3D%2010%3B%0A%20%20px-%3Ebuf%5B1%5D%20%3D%2011%3B%0A%20%20px-%3Ebuf%5B2%5D%20%3D%2012%3B%0A%20%20px-%3Ebuf%5B3%5D%20%3D%2013%3B%0A%20%20px-%3Ebuf%5B4%5D%20%3D%2014%3B%0A%20%20px-%3Ebuf%5B5%5D%20%3D%2015%3B%0A%20%20px-%3Ebuf%5B6%5D%20%3D%2016%3B%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false))
* function pointers aren’t visualized since function code doesn’t appear ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0A//%20function%20pointers%20aren't%20properly%20visualized%0A%23include%20%3Cstdio.h%3E%0A%0Avoid%20fun%28int%20x%29%20%7B%0A%20%20printf%28%22x%20is%20%25d%5Cn%22,%20x%29%3B%0A%7D%0A%20%20%0Aint%20main%28%29%20%7B%0A%20%20void%20%28*fun_ptr%29%28int%29%20%3D%20%26fun%3B%20//%20shows%20up%20as%20poo%20emoji%20since%20there's%20no%20visualization%20for%20function%20fun%0A%20%20%28*fun_ptr%29%2810%29%3B%20//%20...%20but%20the%20call%20actually%20works!%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false))
* long doubles aren’t supported and show up as ‘?’ ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Avoid%20main%28%29%20%7B%0A%20%20//%20long%20doubles%20don't%20display%20properly%20and%20show%20up%20as%20'%3F'%0A%20%20//%20instead%20use%20double%20or%20float,%20which%20both%20work%20fine%3A%0A%20%20long%20double%20ld%20%3D%203.14159%3B%0A%20%20double%20d%20%3D%20ld%3B%0A%20%20float%20f%20%3D%20ld%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false))
* [bit fields](https://en.cppreference.com/w/cpp/language/bit_field) aren’t supported and may crash visualizer
* single-expression / single-line *while* loop may get skipped ([example1](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Aint%20main%28%29%20%7B%0A%20%20int%20x%20%3D%2010%3B%0A%20%20//%20known%20bug%3A%20visualizer%20skips%20over%20steps%20for%20a%20while%281%29%20loop%0A%20%20//%20with%20only%201%20line%20in%20the%20body%3A%0A%20%20while%20%281%29%20%7B%0A%20%20%20%20x%20%2B%3D%201%3B%0A%20%20%7D%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false), [example2](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Aint%20main%28%29%20%7B%0A%20%20int%20x%20%3D%200%3B%0A%20%20//%20known%20bug%3A%20visualizer%20skips%20over%20steps%20for%20a%20while%20loop%0A%20%20//%20where%20the%20body%20is%20on%20the%20same%20line%20as%20the%20loop%20condition%3A%0A%20%20while%20%28x%20%3C%20100%29%20%7Bx%20%2B%3D%201%3B%7D%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false))
* in general, more complex or modern C++ features are *not* supported, such as:
  + exceptions, smart pointers, inline functions, lots of C++11 and newer stuff
  + STL containers and strings (supports only C-style arrays and strings)
  + do *not* rely on visualizer to accurately show inheritance, static members/ methods, virtual methods, public/protected/private modifiers ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0A%0A//%20Class%20hierarchy%3A%0A//%0A//%20%20%20%20%20%20%20%20%20A%0A//%20%20%20%20%20%20%20%20%20%7C%0A//%20C%20%20%20D%20%20%20E%0A//%20%20%5C%20%20%7C%20%20/%0A//%20%20%20%5C%20%7C%20/%0A//%20%20%20%20%20B%0A%0Aclass%20A%20%7B%0Apublic%3A%0A%20%20int%20publicIntA%3B%0Aprotected%3A%0A%20%20int%20protectedIntA%3B%0Aprivate%3A%0A%20%20int%20privateIntA%3B%0A%7D%3B%0A%0Aclass%20C%20%7B%0Apublic%3A%0A%20%20int%20publicIntC%3B%0Aprotected%3A%0A%20%20int%20protectedIntC%3B%0Aprivate%3A%0A%20%20int%20privateIntC%3B%0A%7D%3B%0A%0Aclass%20D%20%7B%0Apublic%3A%0A%20%20int%20publicIntD%3B%0Aprotected%3A%0A%20%20int%20protectedIntD%3B%0Aprivate%3A%0A%20%20int%20privateIntD%3B%0A%7D%3B%0A%0Aclass%20E%20%3A%20public%20A%20%7B%0Apublic%3A%0A%20%20int%20publicIntE%3B%0Aprotected%3A%0A%20%20int%20protectedIntE%3B%0Aprivate%3A%0A%20%20int%20privateIntE%3B%0A%7D%3B%0A%0Aclass%20B%20%3A%20public%20C,%20protected%20D,%20private%20E%20%7B%0Apublic%3A%0A%20%20int%20publicIntB%3B%0Aprivate%3A%0A%20%20int%20privateIntB%3B%0A%7D%3B%0A%0Aint%20main%28%29%20%7B%0A%20%20//%20limitation%3A%20note%20how%20superclass%20member%20variables%0A%20%20//%20are%20NOT%20shown%20in%20the%20visualization%0A%20%20B%20myB%3B%0A%20%20E%20myE%3B%0A%20%20myB.publicIntB%20%3D%20222%3B%0A%20%20myE.publicIntA%20%3D%20555%3B%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=cpp_g%2B%2B9.3.0&rawInputLstJSON=%5B%5D&textReferences=false))
  + C++ destructors on heap objects mess up the visualization ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0A%0A//%20if%20you%20uncomment%20the%20destructor-related%20code,%20the%20visualization%0A//%20gets%20messed%20up%0Aclass%20Computer%20%7B%0Apublic%3A%0A%20%20Computer%28%29%20%7B%0A%20%20%20%20processorspeed%20%3D%200%3B%0A%20%20%7D%0A%20%20%0A%20%20void%20setspeed%20%28int%20p%29%20%7B%0A%20%20%20%20processorspeed%20%3D%20p%3B%0A%20%20%7D%0A%20%20%0A%20%20//~Computer%28%29%20%7B%7D%0A%20%20%0Aprotected%3A%0A%20%20int%20processorspeed%3B%0A%7D%3B%0A%0Aint%20main%28%29%20%7B%0A%20%20Computer*%20heapComputeArray%20%3D%20new%20Computer%5B3%5D%3B%0A%20%20heapComputeArray%5B1%5D.setspeed%281%29%3B%0A%20%20delete%5B%5D%20heapComputeArray%3B%0A%20%20return%200%3B%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=cpp_g%2B%2B9.3.0&rawInputLstJSON=%5B%5D&textReferences=false))
  + rvalue references and move semantics may crash the visualization
* compile-time magic (e.g., preprocessor macros, C++ templates) not visualized

### **JavaScript known limitations**

This visualizer currently supports JavaScript ES6 (running in Node.js v6.0.0).

Here are some things that it does ***not*** support:

* asynchronous and event-driven code
  + including setTimeout, setInterval, etc.
  + promises, async/await
* anything that operates on webpages, such as browser-only objects, DOM manipulation, alert(), prompt(), confirm(), etc.
  + this includes web browser objects like document, window, etc. that exist only in the browser and not on the command-line
  + this also includes trying to import frontend libraries or frameworks (e.g., jQuery, React)
  + remember, this tool is for learning about how the core JavaScript language works, *not* how to build websites or frontends with it
* Date() object
* doesn’t support some advanced features that novices don't usually need to know about, like adding object fields to an array, which can be confusing ([example](http://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Alet%20a%20%3D%20%5B'apples',%202,%20true%5D%3B%20//%20array%0A//%20if%20you%20add%20object%20fields%20to%20arrays%20%28or%20other%20types%29,%20it%20doesn't%20get%20visualized%0Aa.myName%20%3D%20%22Bobby%22%0Aa.myNumber%20%3D%2042%3B%0Aconsole.log%28a,%20a.myName,%20a.myNumber%29%3B&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=js&rawInputLstJSON=%5B%5D&textReferences=false))
* for-loop variables show up *duplicated* in two nested blocks due to how the Node.js JavaScript debugger emits its values ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0A%0A//%20notice%20how%20there%20are%202%20copies%20of%20the%20i%20and%20j%20loop%20variables%20in%20visualizer%3A%0Afor%20%28let%20i%20%3D%200%3B%20i%20%3C%2010%3B%20i%2B%2B%29%20%7B%0A%20%20for%20%28let%20j%20%3D%200%3B%20j%20%3C%20i%3B%20j%2B%2B%29%20%7B%0A%20%20%20%20console.log%28i,%20j%29%3B%0A%20%20%7D%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=js&rawInputLstJSON=%5B%5D&textReferences=false))
* if you have code with side effects or mutations in a [property getter](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/get), it may behave differently since the visualizer calls those getters to produce the execution trace; it’s a good idea never to have side effects or mutations in getters
* if your code runs for too long or has an infinite loop, the server may crash without showing you a partial trace of what steps led to that crash
* closures and parent frames may sometimes be inaccurate since Node.js does not always expose this data to tracing tools in a precise way

### 

### **Java known limitations**

[**Summary doc of Java visualizer features**](https://pythontutor.com/articles/java-visualizer.html)

The Java visualizer was originally made by external developers [David Pritchard](https://github.com/daveagp) and Will Gwozdz. It runs Java 8, which is a stable version that is well-suited for introductory and intermediate-level Java courses. It also pre-imports some libraries from Princeton such as [StdIn](http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdIn.html), [StdOut](http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdOut.html), most other [stdlib libraries](http://introcs.cs.princeton.edu/java/stdlib), [Stack](http://introcs.cs.princeton.edu/java/43stack/Stack.java.html), [Queue](http://introcs.cs.princeton.edu/java/43stack/Queue.java.html), and [ST](http://introcs.cs.princeton.edu/java/44st/ST.java.html).

Known limitations:

* do not use the package keyword or else the visualizer will crash ([example](https://pythontutor.com/render.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Apackage%20MyHomework%3B%20//%20do%20not%20include%20the%20'package'%20keyword%0A%0Apublic%20class%20Example%20%7B%0A%20%20public%20static%20void%20main%28String%5B%5D%20args%29%20%7B%0A%20%20%20%20System.out.println%28%22Hello%20world!%22%29%3B%0A%20%20%7D%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=java&rawInputLstJSON=%5B%5D&textReferences=false)); there is no need to declare package
* do not spawn Threads or else the visualizer will crash; it does not support [multi-threaded execution](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html)
* visualizer doesn’t display the declared types of variables ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Aclass%20Animal%20%7B%7D%0Aclass%20Duck%20extends%20Animal%20%7B%7D%3B%0Aclass%20Cat%20extends%20Animal%20%7B%7D%3B%0A%0Apublic%20class%20Example%20%7B%0A%20%20public%20static%20void%20main%28String%5B%5D%20args%29%20%7B%0A%20%20%20%20//%20Python%20Tutor%20doesn't%20show%20that%20the%20*declared%20types*%0A%20%20%20%20//%20of%20a1,%20a2,%20a3%20are%20all%20'Animal'%0A%20%20%20%20Animal%20a1%20%3D%20new%20Animal%28%29%3B%0A%20%20%20%20Animal%20a2%20%3D%20new%20Duck%28%29%3B%0A%20%20%20%20Animal%20a3%20%3D%20new%20Cat%28%29%3B%0A%20%20%7D%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=java&rawInputLstJSON=%5B%5D&textReferences=false))
* doesn’t display public/protected/private modifiers on instance variables ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Aclass%20Animal%20%7B%0A%20%20public%20int%20age%3B%0A%20%20protected%20String%20species%3B%0A%20%20private%20boolean%20isCarnivore%3B%0A%0A%20%20public%20void%20setCarnivoreStatus%28boolean%20c%29%20%7BisCarnivore%3Dc%3B%7D%3B%0A%7D%0A%0Apublic%20class%20Example%20%7B%0A%20%20public%20static%20void%20main%28String%5B%5D%20args%29%20%7B%0A%20%20%20%20//%20Python%20Tutor%20doesn't%20show%20public/private/protected%20modifiers%3A%0A%20%20%20%20Animal%20a1%20%3D%20new%20Animal%28%29%3B%0A%20%20%20%20a1.age%20%3D%2012%3B%0A%20%20%20%20a1.species%20%3D%20%22canine%22%3B%0A%20%20%20%20a1.setCarnivoreStatus%28true%29%3B%0A%20%20%7D%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=java&rawInputLstJSON=%5B%5D&textReferences=false))
* Java Collections data structures like ArrayList and LinkedList aren't visualized (built-in arrays are visualized, though) ([example](https://pythontutor.com/visualize.html#code=//%20%5Bofficial%20Python%20Tutor%20example%5D%0Aimport%20java.util.*%3B%0A%0Apublic%20class%20Example%20%7B%0A%20%20public%20static%20void%20main%28String%5B%5D%20args%29%20%7B%0A%20%20%20%20//%20regular%20array%20renders%20fine%0A%20%20%20%20String%20a%5B%5D%20%3D%20%7B%22Apple%22,%20%22Banana%22,%20%22Coconut%22%7D%3B%0A%20%20%20%20//%20Python%20Tutor%20doesn't%20show%20the%20internal%20contents%20of%20ArrayList,%20LinkedList,%20etc.%0A%20%20%20%20ArrayList%20arrList%20%3D%20new%20ArrayList%3CString%3E%28%29%3B%0A%20%20%20%20LinkedList%20ll%20%3D%20new%20LinkedList%3CString%3E%28%29%3B%0A%20%20%20%20arrList.add%28%22dog%22%29%3B%0A%20%20%20%20arrList.add%28%22cat%22%29%3B%0A%20%20%20%20ll.add%28%22horse%22%29%3B%0A%20%20%20%20ll.add%28%22donkey%22%29%3B%0A%20%20%20%20System.out.println%28arrList%29%3B%0A%20%20%20%20System.out.println%28ll%29%3B%0A%20%20%7D%0A%7D&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=java&rawInputLstJSON=%5B%5D&textReferences=false))
* taking text input via terminal/console is not supported. e.g., code like this will not work in the visualizer: Scanner scanner = new Scanner(System.in);
* command-line arguments are not supported
* to access built-in Stack and Queue from the Java standard library, you must write something like: import java.util.Stack. This won't work: import java.util.\*;

## Why are there ads?

## 

## **Python Tutor is a high-traffic website that requires continuous funding to pay for server costs to keep it online.**

This website is a nonprofit academic university project which gets a substantial amount of usage. To keep our servers running, it requires a stable source of funding. Since we want to **keep Python Tutor free for all users**, we use Google AdSense banner ads.

This mechanism is *completely automated*, and we cannot select or monitor these ads. Thus, we are not responsible for the contents of these ads, and we cannot predict which ads will be displayed for you since it may depend on your web browsing history.

Alternatively, you can make a [**small monthly donation**](https://docs.google.com/forms/d/e/1FAIpQLSeVehBoLGJhbB9GGSh5Bj8AWa0rF5NCMjLCtuA3rhaEJlsRlg/viewform?usp=sf_link) to get an ad-free experience with future premium features. Thanks for your support!

(end of documentation)