

# Scope

## Lecture 5

Centre for Data Science, ITER  
Siksha 'O' Anusandhan (Deemed to be University),  
Bhubaneswar, Odisha, India.

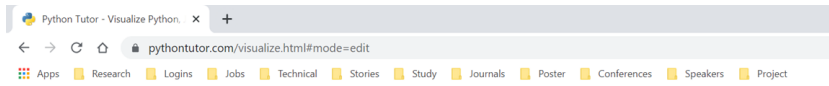


# Contents

- 1 Introduction
- 2 del operator
- 3 Namespaces
- 4 Scope

# Introduction: Visualization of Example-I in Python tutor

- Each object in Python is assigned a unique identifier that can be accessed using the function id.
- Visit the following website [www.pythontutor.com](http://www.pythontutor.com).
- Click on Edit this code and write the code.
- Click on Visualize Execution.



Write code in Python 3.6

```
1 def main():
2     a=5
3     print('a=',a,'id(a)=',id(a))
4     b=3+2
5     print('b=',b,'id(b)=',id(b))
6     a=7
7     print('a=',a,'id(a)=',id(a))
8     print('b=',b,'id(b)=',id(b))
9 if __name__ == '__main__':
10     main()
```

Visualize Execution

Live Programming Mode

hide exited frames [default]

inline primitives, don't nest objects [default]

draw pointers as arrows [default]

# Visualization of Example-I in Python tutor (Continue...)

- After clicking on Visualize Execution, a iPrint output box will appear at the right top.

The screenshot displays the Python Tutor interface. The browser address bar shows the URL `pythontutor.com/visualize.html#mode=display`. Below the address bar is a navigation menu with links: Apps, Research, Logins, Jobs, Technical, Stories, Study, Journals, Poster, Conferences, Speakers, and Projects. A link to "Get live help in the Python Discord chat" is also present.

The main content area is titled "Python 3.6 (known limitations)". It contains a Python script with 11 lines of code:

```
1 def main():
2     a=5
3     print('a=',a,'id(a)=',id(a))
4     b=3+2
5     print('b=',b,'id(b)=',id(b))
6     a=7
7     print('a=',a,'id(a)=',id(a))
8     print('b=',b,'id(b)=',id(b))
9
10 if __name__=='__main__':
11     main()
```

Below the code, there is a legend:

- line that just executed
- next line to execute

A progress bar is shown at the bottom of the code area, with buttons for navigation: << First, < Prev, Next >, and Last >>. The current step is indicated as "Step 1 of 12". A link "Customize visualization" is also available.

On the right side, there is a "Print output (drag lower right corner to resize)" box, which is currently empty. Below it are two tabs: "Frames" and "Objects".

# Visualization of Example-I in Python tutor (Continue...)

- To start visualization, we click <Next >. On encountering the main function definition, the global frame lists the identifier main as shown in the Figure.
- The red arrow marks the next line to be executed, and the green arrow marks the line just executed.

The screenshot displays the Python Tutor web interface. The browser tab is titled "Python Tutor - Visualize Python." and the URL is "pythontutor.com/visualize.html#mode=display". A navigation bar contains links for Apps, Research, Logins, Jobs, Technical, Stories, Study, Journals, Poster, Conferences, Speakers, and Projects. A link for "Get live help in the Python Discord chat" is also present.

The main area shows a Python 3.6 code snippet (with a link to "known limitations"):

```
1 def main():
2     a=5
3     print('a=',a,'id(a)=',id(a))
4     b=3+2
5     print('b=',b,'id(b)=',id(b))
6     a=7
7     print('a=',a,'id(a)=',id(a))
8     print('b=',b,'id(b)=',id(b))
9
10 if __name__=='__main__':
11     main()
```

Below the code, a legend indicates that a green arrow (→) marks the line just executed and a red arrow (→) marks the next line to execute. A scrollbar and navigation buttons (<< First, < Prev, Next >, Last >>) are at the bottom of the code area.

On the right, the "Print output" section is empty. Below it, the "Frames" and "Objects" panels show the current state:

- Frames:** A box labeled "Global frame" contains the identifier "main".
- Objects:** A box labeled "function" contains the identifier "main()".

A blue arrow points from the "main" identifier in the Global frame to the "main()" function object in the Objects panel.

At the bottom right, the text "Step 2 of 12" is displayed.

# Visualization of Example-I in Python tutor (Continue...)

- Now clicking <Next>, it executes the if statement. Clicking <Next> again executes the call to the function main and the visualizer shows the frame for the main function.
- Clicking <Next>, it moves the next line pointer to line 2, and its execution shows the creation of int object 5 having name a. Later, clicking <Next> shows the output of the execution of line 3 in <Print output> box.

Python Tutor - Visualize Python, x +

pythonututor.com/visualize.html#mode=display

Apps Research Logins Jobs Technical Stories Study Journals Poster Conferences Speakers Proje

[Get live help](#) in the [Python Discord](#) chat

Python 3.6  
([known limitations](#))

```
1 def main():
2     a=5
3     print('a=',a,'id(a)=',id(a))
4     b=3+2
5     print('b=',b,'id(b)=',id(b))
6     a=7
7     print('a=',a,'id(a)=',id(a))
8     print('b=',b,'id(b)=',id(b))
9
10 if __name__=='__main__':
11     main()
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

Print output (drag lower right corner to resize)

```
a= 5 id(a)= 140426606966816
```

Frames Objects

Global frame  
main → function main()

main  
a 5

6 / 35

# Visualization of Example-I in Python tutor (Continue...)

- Next click executes line 4 and the name b is mapped to the int object 5 created earlier.
- Further click executes line 5 and the output appears in the <Print output> box.

The screenshot displays the Python Tutor web application interface. At the top, the browser address bar shows the URL `pythontutor.com/visualize.html#mode=display`. Below the address bar is a navigation menu with links for Apps, Research, Logins, Jobs, Technical, Stories, Study, Journals, Poster, Conferences, Speakers, and Projects. A link for "Get live help in the Python\_Discord chat" is also present.

The main content area is divided into three sections:

- Code Editor:** Displays Python 3.6 code with the following lines:

```
1 def main():
2     a=5
3     print('a=',a,'id(a)=',id(a))
4     b=3+2
5     print('b=',b,'id(b)=',id(b))
6     a=7
7     print('a=',a,'id(a)=',id(a))
8     print('b=',b,'id(b)=',id(b))
9
10 if __name__=='__main__':
11     main()
```

Line 5 is highlighted with a green arrow, indicating it is the current line of execution. Line 6 is marked with a red arrow, indicating the next line to be executed. A legend below the code explains these markers: a green arrow for "line that just executed" and a red arrow for "next line to execute".
- Print output:** A box on the right shows the output of the program:

```
a= 5 id(a)= 140426606966816
b= 5 id(b)= 140426606966816
```
- Frames and Objects:** A diagram showing the state of memory. It includes a "Global frame" containing a variable "main" which points to a "function main()". Below this, a table for the "main" frame shows the current state of variables:

Variable	Value
a	5
b	5

At the bottom of the interface, there is a "Customize visualization" link, a "Step 9 of 12" indicator, and navigation buttons: "<< First", "< Prev", "Next >", and "Last >>".

# Visualization of Example-I in Python tutor (Continue...)

- Clicking <Next> executes line 6, resulting in creation of a new int object 7 and its mapping to a.
- Further clicks execute lines 7 and 8 and the output appears in the <Print output> box.
- Next click shows return value None associated with the function main as it does not return any value.

The screenshot displays the Python Tutor interface for Python 3.6. The code being executed is as follows:

```
Python 3.6
(known limitations)

1 def main():
2     a=5
3     print('a=',a,'id(a)=' ,id(a))
4     b=3+2
5     print('b=',b,'id(b)=' ,id(b))
6     a=7
7     print('a=',a,'id(a)=' ,id(a))
8     print('b=',b,'id(b)=' ,id(b))
9
10 if __name__=='__main__':
11     main()
```

Navigation controls at the bottom include: << First, < Prev, Next >, Last >>. A slider indicates the current execution position at line 8.

Legend:

- line that just executed
- next line to execute

**Print output (drag lower right corner to resize)**

```
a= 5 id(a)= 140426606966816
b= 5 id(b)= 140426606966816
a= 7 id(a)= 140426606966880
b= 5 id(b)= 140426606966816
```

**Frames**

Frames	Objects
Global frame	
main	function main()

**main**

a	7
b	5
Return value	None



# Visualization of Example-I in Python tutor (Continue...)

- Recall that when this program script is executed, Python makes a note of the definition of the function `main` in the global frame.
- Next, on encountering the `if` statement, Python checks whether the name of the current module is `__main__`.
- This being true, the expression `__name__ == '__main__'` evaluates as `True`, and the function `main` gets invoked.
- Next, the statements in the `main` function are executed in a sequence.
- Execution of line 2 creates an `int` object 5 and assigns it the name `a`. This object has a unique object id but can have multiple names as the execution of the script proceeds.
- For example, execution of the statement

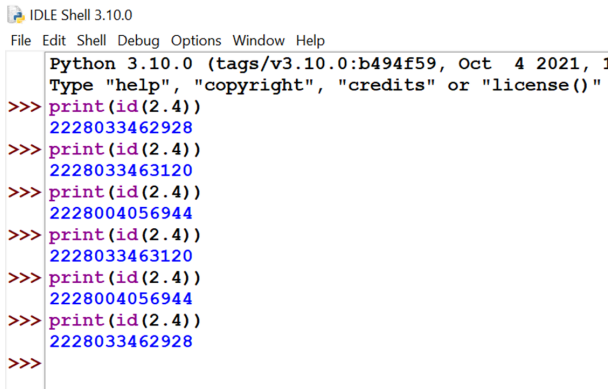
$$b = 3 + 2 \tag{1}$$

in line 4 does not generate a new object, but only associates the name `b` to the `int` object 5 created earlier

- Now, `a` and `b` have the same object id. However, execution of line 6 creates an `int` object 7 and associates it with the name `a`.
- The name `b` continues to be associated with `int` object 5 created earlier.

# Object ID for different data types

- The general principle is expressed by saying that Python caches or interns small integer objects (typically, up to 100) for future use. The same may not hold for other forms of data.

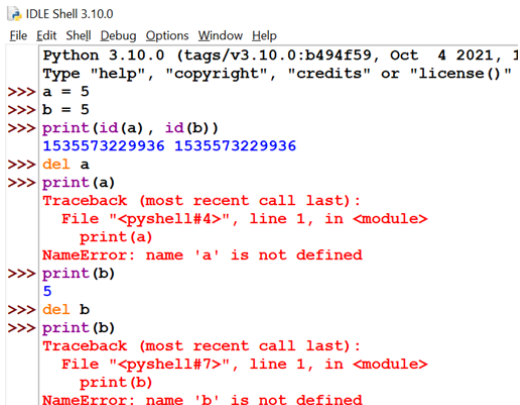


```
IDLE Shell 3.10.0
File Edit Shell Debug Options Window Help
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 1
Type "help", "copyright", "credits" or "license()"
>>> print(id(2.4))
2228033462928
>>> print(id(2.4))
2228033463120
>>> print(id(2.4))
2228004056944
>>> print(id(2.4))
2228033463120
>>> print(id(2.4))
2228004056944
>>> print(id(2.4))
2228033462928
>>>
```

- Note that the first three instructions create new objects.
- However, subsequent instructions sometimes used the objects created earlier.

# del operator

- It makes a name (i.e. the association between the name and the object) undefined.



```
IDLE Shell 3.10.0
File Edit Shell Debug Options Window Help
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 1
Type "help", "copyright", "credits" or "license()"
>>> a = 5
>>> b = 5
>>> print(id(a), id(b))
1535573229936 1535573229936
>>> del a
>>> print(a)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print(a)
NameError: name 'a' is not defined
>>> print(b)
5
>>> del b
>>> print(b)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print(b)
NameError: name 'b' is not defined
```

- The first statement creates an int object 5 and binds it to name a.
- The second statement does not create a new object.
- It binds the same object to name b and thus creates another reference to int object a.

# Visualization of del operator in Python tutor

- The first statement creates an int object 5 and binds it to name a.
- The second statement does not create a new object.
- It binds the same object to name b and thus creates another reference to int object a.

Python Tutor - Visualize Python. x +

← → ↺ 🏠 🔒 pythontutor.com/visualize.html#mode=display

Apps Research Logins Jobs Technical Stories Study Journals Poster Conferences Speak

[Get live help](#) in the [Python Discord](#) chat

Python 3.6  
([known limitations](#))

```
1 a=5
2 b=5
3 print(id(a),id(b))
4 print(a)
5 del a
6 print(b)
7 del b
8 print(b)
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

Step 3 of 8

[Customize visualization](#)

Print output (drag lower right corner to resize)

Frames Objects

Global frame

b	5
a	5

# Visualization of del operator (continued)

- Python keeps a count of the number of references to an object.
- When the statement `del a` is executed, it reduces the reference count of int object 5 from 2 to 1 and removes the binding of name `a` to int object 5.
- Thus, an attempt to access `a` now yields an error.

The screenshot shows the Python Tutor interface. The code being executed is:

```
1 a=5
2 b=5
3 print(id(a),id(b))
4 print(a)
5 del a
6 print(b)
7 del b
8 print(b)
```

Line 5, `del a`, is highlighted with a green arrow, indicating it is the line that just executed. Line 6, `print(b)`, is highlighted with a red arrow, indicating it is the next line to execute.

The output window shows the print output (drag lower right corner to resize):

```
140366912186400 140366912186400
5
```

The Frames panel shows the Global frame with the variable `b` pointing to the value `5`.

Step 6 of 8

# Visualization of del operator (continued)

- When the statement `del b` is executed, it reduces the reference count of int object 5 from 1 to 0, and removes the binding of name `b` to int object 5.
- Thus, an attempt to access `b` now yields an error.

The screenshot shows the Python Tutor interface with the following components:

- Browser Header:** "Python Tutor - Visualize Python, x +"
- Address Bar:** "pythontutor.com/visualize.html#mode=display"
- Navigation:** Back, Forward, Refresh, Home icons.
- Tags:** Apps, Research, Logins, Jobs, Technical, Stories, Study, Journals, Poster, Conferences, Speake.
- Help:** "Get live help in the Python Discord chat"
- Code Editor:**

```
Python 3.6
(known limitations)
1 a=5
2 b=5
3 print(id(a),id(b))
4 print(a)
5 del a
6 print(b)
→ 7 del b
→ 8 print(b)
```

[Edit this code](#)
- Legend:**
  - line that just executed
  - next line to execute
- Execution Progress:** A slider bar and buttons: "<< First", "< Prev", "Next >", "Last >>". Below the buttons is "Step 8 of 8".
- Output Panel:**

Print output (drag lower right corner to resize)

```
140366912186400 140366912186400
5
5
```

Frames      Objects
- Footer:** [Customize visualization](#)

# Visualization of Example-II in Python tutor

- Write a program for calculating the percentage of a student in a subject.

The screenshot shows the Python Tutor web interface. The browser tab is titled "Python Tutor - Visualize Python, ..." and the address bar shows the URL "pythontutor.com/visualize.html#mode=display". Below the address bar, there are navigation links for "Apps", "Research", "Logins", "Jobs", "Technical", "Stories", "Study", "Journals", "Poster", and "Confere". A link to "Get live help in the Python Discord chat" is also present.

The main content area displays the Python code for calculating a student's percentage. The code is as follows:

```
Python 3.6
(known limitations)

1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13     main()
```

Below the code, there is a link to "Edit this code".

On the right side of the interface, there are two tabs: "Frames" and "Objects".

At the bottom of the interface, there is a legend indicating that a green arrow points to the line that just executed and a red arrow points to the next line to execute. Below the legend is a progress bar and a set of navigation buttons: "<< First", "< Prev", "Next >", and "Last >>".

The bottom of the interface shows "Step 1 of 5" and a link to "Customize visualization".

# Visualization of Example-II (continued)

- We are about to execute the first of the five steps in the script.
- These five steps are as follows:
  - Definition of function percent(marks, maxMarks)
  - Definition of function main()
  - if statement
  - Invoking the function main()
  - Execution of function main()



# Visualization of Example-II (continued)

- On clicking <Next>, step 1 is executed and we see the function percent in the global frame.

Python Tutor - Visualize Python, x +

python tutor.com/visualize.html#mode=display

Apps Research Logins Jobs Technical Stories Study Journals Poster Conferences Speakers Proc

Python 3.6  
([known limitations](#))

```
1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13     main()
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

Step 2 of 5

Frames: Global frame (percent) → Objects: function percent(marks, maxMarks)

Customize visualization

# Visualization of Example-II (continued)

- On clicking <Next>, step 1 is executed and we see the function percent in the global frame.
- On execution of next step, the function main is also shown in the global frame.

Python Tutor - Visualize Python, . x +

← → ↺ 🏠 pythonthututor.com/visualize.html#mode=display

Apps Research Logins Jobs Technical Stories Study Journals Poster Conferences Speakers Prc

Python 3.6  
([known limitations](#))

```
→ 1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
→ 5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13     main()
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

<< First < Prev Next > Last >>

Step 2 of 5

Customize visualization

Frames      Objects

Global frame  
percent

function  
percent(marks, maxMarks)

18/35

# Visualization of Example-II (continued)

- When step 3 is executed, the condition `if __name__=='__main__'` evaluates as True.
- Thus, in step 4 the function `main` is invoked.

Python Tutor - Visualize Python, x +

← → ↺ 🏠 🔒 pythontutor.com/visualize.html#mode=display

📱 Apps 📁 Research 📁 Logins 📁 Jobs 📁 Technical 📁 Stories 📁 Study 📁 Journals 📁 Poster 📁 Conferences 📁 Speakers 📁 Pr

Python 3.6  
([known limitations](#))

```
1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
➡ 5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
➡ 12 if __name__=='__main__':
13     main()
```

[Edit this code](#)

➡ line that just executed

➡ next line to execute

Frames

Objects

Global frame  
percent  
main

function  
percent(marks, maxMarks)

function  
main()

<< First

< Prev

Next >

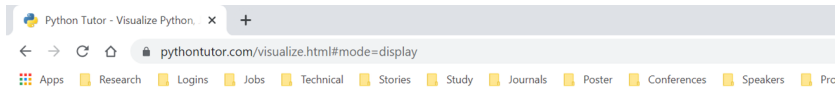
Last >>

Step 3 of 5

19/35

# Visualization of Example-II (continued)

- Next click executes the call to function main and the visualizer shows the function main among frames.



[Get live help](#) in the [Python Discord](#) chat

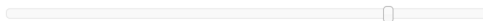
Python 3.6  
([known limitations](#))

```
1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13     main()
```

[Edit this code](#)

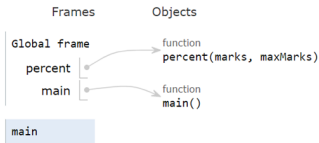
→ line that just executed

→ next line to execute



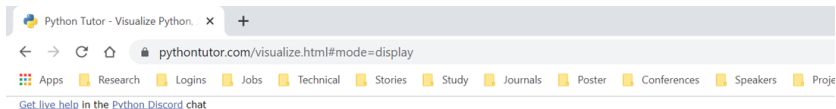
<< First < Prev Next > Last >>

Step 5 of 5



# Visualization of Example-II (continued)

- Next click yields a message that prompts the user to enter total marks in <Input Box> and hit the Submit button.
- This input message along with the values entered as input are also shown in <Print output>.
- At this stage, we enter 500 as total marks, and click Submit.

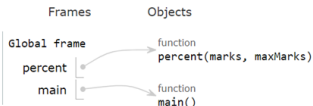


Python 3.6  
(known limitations)

```
1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13     main()
```

[Edit this code](#)

→ line that just executed  
→ next line to execute



# Visualization of Example-II (continued)

- Now the execution is resulting in creation of an instance of float object 500.0.
- This object is named as maxMarks.
- Then next click executes again prompting for marks obtained (say, 450).

Python Tutor - Visualize Python. x +

python tutor.com/visualize.html#mode=display

Apps Research Logins Jobs Technical Stories Study Journals Poster Conferences Speakers Project

Python 3.6  
([known limitations](#))

```
1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13    main()
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

Enter user input:

Enter marks obtained:

Print output (drag lower right corner to resize)

Enter total marks: 500

Frames      Objects

Global frame

- percent → function percent(marks, maxMarks)
- main → function main()

main

maxMarks 500.0

# Visualization of Example-II (continued)

- As before, an instance of float object 450.0 is created, and named as marks.
- The control moves the next line.

Python Tutor - Visualize Python. x +

← → ↺ 🏠 🔒 pythontutor.com/visualize.html#mode=display

Apps

Research

Logins

Jobs

Technical

Stories

Study

Journals

Poster

Conferences

Speakers

Project

Python 3.6  
([known limitations](#))

```
1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13    main()
```

[Edit this code](#)

→ line that just executed

→ next line to execute

<< First

< Prev

Next >

Last >>

Enter user input:

Enter marks obtained:

Print output (drag lower right corner to resize)

Enter total marks: 500

Frames

Objects

Global frame

- percent → function percent(marks, maxMarks)
- main → function main()

main

- maxMarks 500.0

[Customize visualization](#)

# Visualization of Example-II (continued)

- Clicking <Next> executes the call to the function percent and the visualizer shows the function percent among frames.
- Note that the parameters marks and maxMarks are mapped to float objects created earlier.

Python Tutor - Visualize Python. x +

← → ↺ 🏠 python tutor.com/visualize.html#mode=display

📱 Apps 📁 Research 📁 Logins 📁 Jobs 📁 Technical 📁 Stories 📁 Study 📁 Journals 📁 Poster 📁 Conferences 📁 Speakers 📁 Project

Python 3.6  
([known limitations](#))

→ 1 def percent(marks,maxMarks):

→ 2     percentage = (marks/maxMarks)\*100

3     return percentage

4

5 def main():

6     # To compute percentage

7     maxMarks = float(input('Enter total marks: '))

8     marks = float(input('Enter marks obtained: '))

9     percentage = percent(marks,maxMarks)

10     print('Percentage is: ',percentage)

11

12 if \_\_name\_\_=='\_\_main\_\_':

13     main()

[Edit this code](#)

→ line that just executed  
→ next line to execute

<< First < Prev Next > Last >>

Step 10 of 14

[Customize visualization](#)

Print output (drag lower right corner to resize)

Enter total marks: 500  
Enter marks obtained: 450

Frames      Objects

Global frame  
percent  
main

function  
percent(marks, maxMarks)  
function  
main()

main  
maxMarks 500.0  
marks 450.0

percent  
marks 450.0  
maxMarks 500.0

24/35



# Visualization of Example-II (continued)

- On the next click, Python creates a return value i.e. the float object (to be returned to the main function) 90.0 which was created earlier.

Python Tutor - Visualize Python, x +

← → ↺ 🏠 pythontutor.com/visualize.html#mode=display

Apps Research Logins Jobs Technical Stories Study Journals Poster Conferences Speakers Project

```
Python 3.6
(known limitations)

1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13    main()
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

<< First < Prev Next > Last >>

Step 11 of 14

[Customize visualization](#)

Print output (drag lower right corner to resize)

```
Enter total marks: 500
Enter marks obtained: 450
```

Frames      Objects

Global frame

- percent → function percent(marks, maxMarks)
- main → function main()

main

maxMarks	500.0
marks	450.0

percent

marks	450.0
maxMarks	500.0
percentage	90.0

# Visualization of Example-II (continued)

- The next click returns the value 90.0 from the function percent by associating it with the variable percentage of function main.
- Note that the variable percentage of the function main now maps to the float object 90.0.

Python Tutor - Visualize Python, x +

python tutor.com/visualize.html#mode=display

Apps Research Logins Jobs Technical Stories Study Journals Poster Conferences Speakers Project

Python 3.6  
(known limitations)

```
1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13     main()
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

Step 12 of 14

Print output (drag lower right corner to resize)

```
Enter total marks: 500
Enter marks obtained: 450
```

Frames      Objects

Global frame

- percent → function percent(marks, maxMarks)
- main → function main()

main

maxMarks	500.0
marks	450.0

percent

marks	450.0
maxMarks	500.0
percentage	90.0
Return value	90.0

[Customize visualization](#)

# Visualization of Example-II (continued)

- The next two clicks shows return value None associated with the function main() as it does not return any value.

Python Tutor - Visualize Python. x +

python tutor.com/visualize.html#mode=display

Apps Research Logins Jobs Technical Stories Study Journals Poster Conferences Speakers Project

Python 3.6  
([known limitations](#))

```
1 def percent(marks,maxMarks):
2     percentage = (marks/maxMarks)*100
3     return percentage
4
5 def main():
6     # To compute percentage
7     maxMarks = float(input('Enter total marks: '))
8     marks = float(input('Enter marks obtained: '))
9     percentage = percent(marks,maxMarks)
10    print('Percentage is: ',percentage)
11
12 if __name__=='__main__':
13     main()
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

Step 14 of 14

Print output (drag lower right corner to resize)

```
Enter total marks: 500
Enter marks obtained: 450
Percentage is: 90.0
```

Frames Objects

Global frame

percent → function percent(marks, maxMarks)

main → function main()

main

maxMarks	500.0
marks	450.0
percentage	90.0
Return value	None

[Customize visualization](#)

# Namespaces

- As the term suggests, a namespace is a space that holds some names.
- A namespace defines a mapping of names to the associated objects.
- In Python, a module, class, or function defines a namespace.
- Names that appear in global frame (usually outside of the definition of classes, functions, and objects) are called global names and collectively they define the namespace called global namespace.
- The names introduced in a class or function are said to be local to it.
- The region in a script in which a name is accessible is called its scope.
- Thus, the scope of a name is resolved in the context of the namespace in which it is defined.

# An example

- For example, each of the functions `f1` and `f2` defined in a script may have the name `x`.
- The variable `x` defined in function `f1` may refer to an object of type different from that of the object associated with variable `x` in the function `f2`.
- A namespace maps names to objects.
- Being an object-oriented language, definitions of functions and classes are also examples of objects.

# Scope

- The scope rules for names in Python are often summarized as LEGB rule.
- LEGB stands for local, enclosing, global, and built in.
- All names defined within the body of a function are local to it.
- Function parameters are also considered local.
- If a name is not locally found, Python recursively searches for its definition in an enclosing scope.
- Names defined in the Python script but usually outside of any function or class definition are called global.
- Python defines some built-in names such as len and abs, which can be accessed from anywhere in a program.

# Examples

- Example1: Note that as the variable `a` has global scope, it is accessible in function `f`.

```
1  a = 4
2  def f():
3      print('global a: ', a)
4  f()
```

- Example 2: Note that the name `a` introduced in line 3 in the function `f` is local to it and has associated value 5. Thus defining the value of name `a` in the function `f`, does not affect the value of the global name `a`.

```
1  a = 4.2
2  def f():
3      a = 5
4      print('local a: ', a)
5  f()
6  print('global a: ', a)
```

# Examples (continued)

- Example 3: In this example, during execution of function g, when the variable a is to be accessed, Python looks for it in the local scope of function g, as it is not defined in the body of function g, it is searched in the next enclosing scope, i.e., the scope of function f, where the variable a is indeed defined, and therefore the value 5 of the variable a in function f gets bound to the occurrence of variable a in the function g.

```
1  a = 6
2  def f():
3      a = 5
4      def g():
5          b = a
6          print('inside function g, b: ', b)
7      g()
8  f()
```



# Examples (continued)

- Example 4: In this example, the variable `a` is defined in the body of inner function `g`. When we attempt to access the name `a` in line 5 of the outer function `f`, Python looks for its definition first inside the body of function `f`, as there is no definition of `a` in the function `f`, it looks for definition of `a` in the next available enclosing scope, which is the global scope. Again, there is no definition of the name `a` in global name space, and hence the error message is printed.

```
1  def f():
2      def g():
3          a = 5
4      g()
5      print('in outer function g, a =', a)
6  f()
```

# Conclusions

- Python visualizer is an online tool for visualizing the execution of Python code.
- A namespace defines a mapping of names to the associated objects.
- Names that appear in global frame outside of the definition of classes, functions, and objects are called global names, and collectively they define the namespace called global namespace.
- The names introduced in a class, or function are said to be local to it.
- The region in a script in which a name is accessible is called its scope.
- The scope rules for names in Python are often summarized as LEGB rule
- If a name is not locally defined, Python recursively searches for its definition in an enclosing scope.
- Python defines some built-in names such as len and abs which can be accessed from anywhere in a program.

# References

- [1] Python Programming: A modular approach by Taneja Sheetal, and Kumar Naveen, *Pearson Education India, Inc.*, 2017.