

# Programming-IPython

---

**CPSC 217: Introduction to Computer Science for Multidisciplinary  
Studies I  
Winter 2023**

Jonathan Hudson, Ph.D.  
Instructor  
Department of Computer Science  
University of Calgary

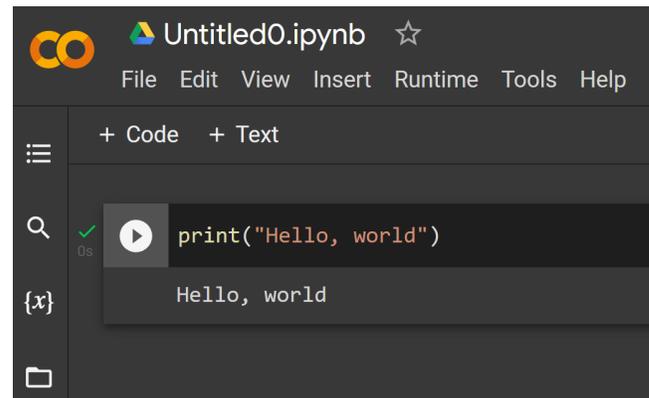
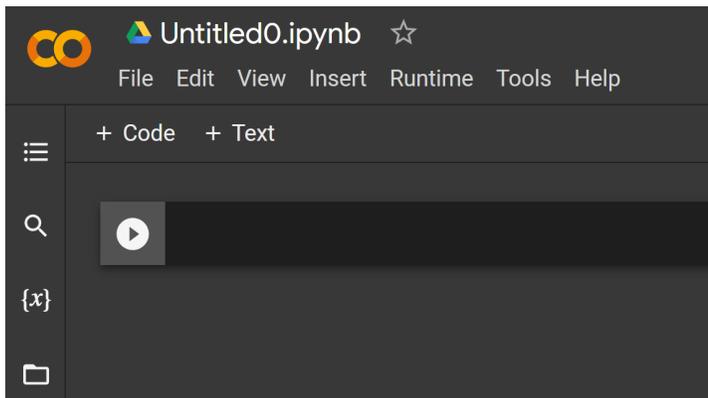
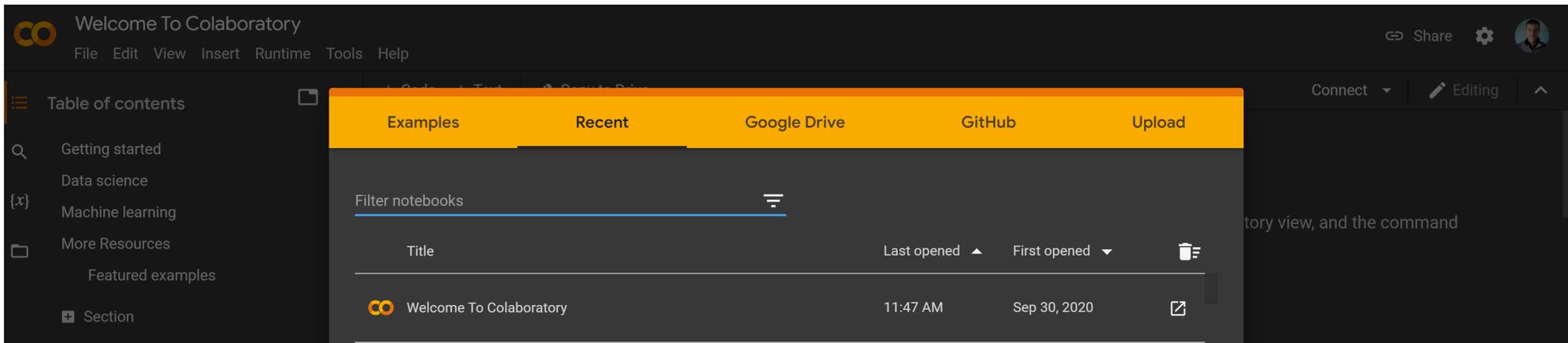
*January 9, 2023*

*Copyright © 2023*



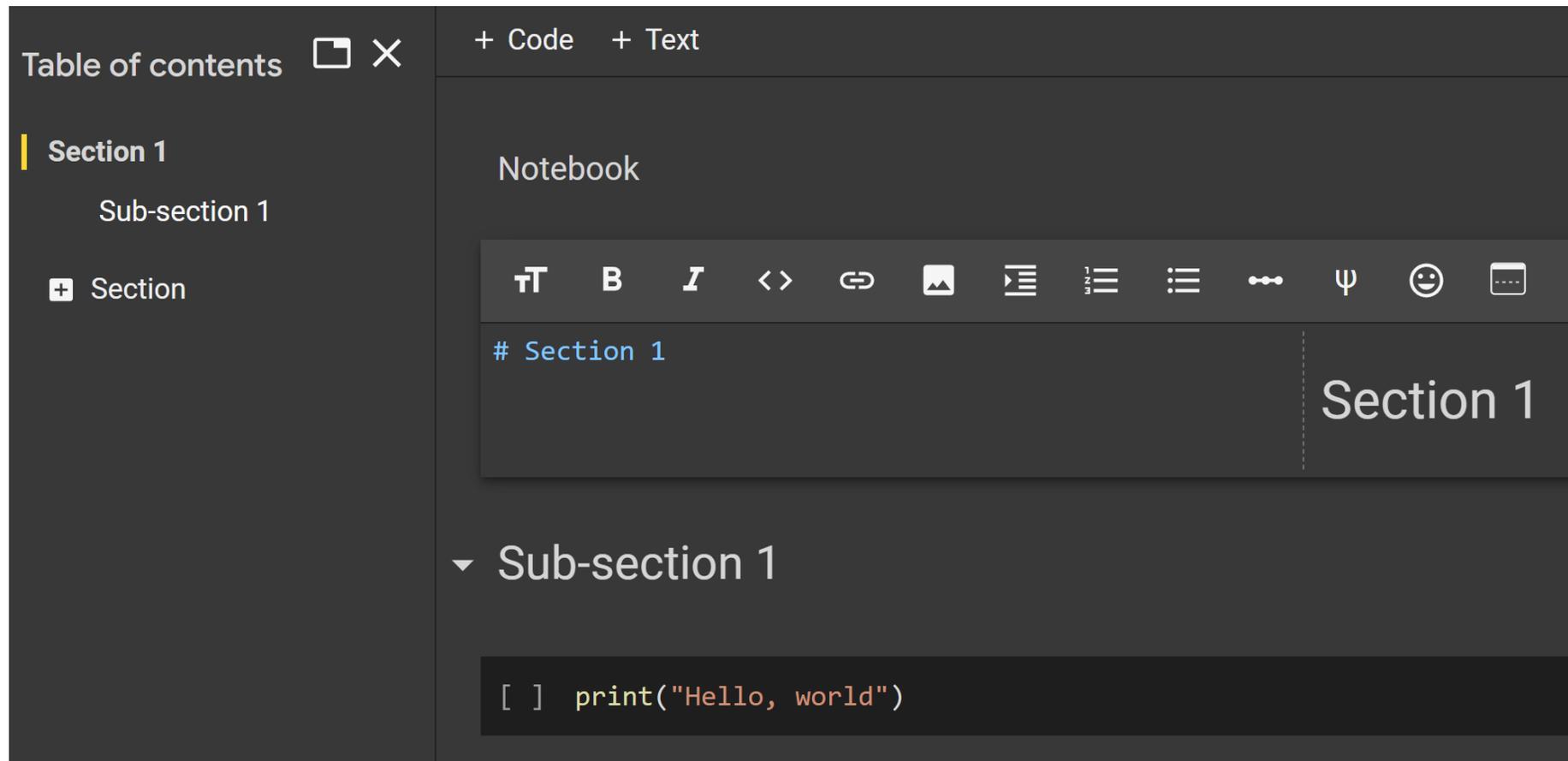
# Beginner?

Just use **Google Colaboratory!** (web-based version of **Python Jupyter** notebook)



# IPython (markdown language)

<https://www.markdownguide.org/cheat-sheet/>

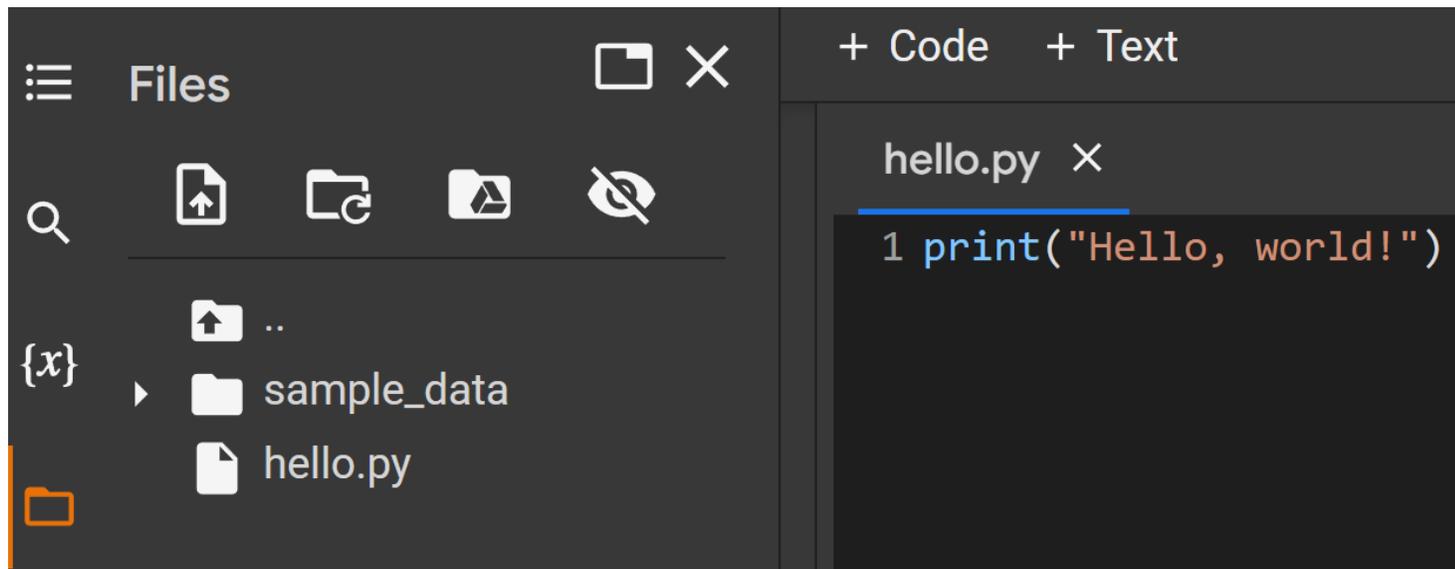


The screenshot displays an IPython notebook interface. On the left, a 'Table of contents' sidebar lists 'Section 1' (highlighted), 'Sub-section 1', and a '+ Section' button. The main area is titled 'Notebook' and features a toolbar with icons for text formatting (bold, italic, code), linking, image insertion, list creation, and other functions. Below the toolbar, a code cell contains the markdown header `# Section 1`, which is rendered as 'Section 1' in the notebook's output. A second code cell below it contains the Python code `[ ] print("Hello, world")`.

# IPython (files)

---

- Supports additional file creation and management (also Google Drive linking)

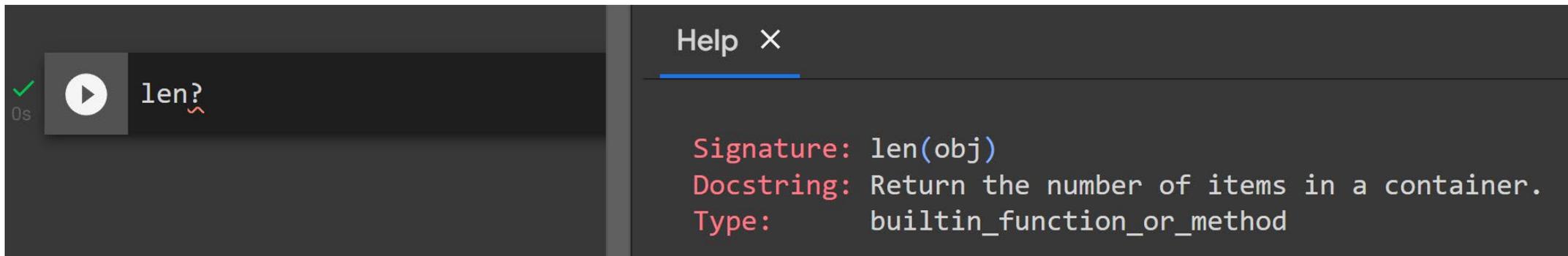


- By default saved in **Google Drive**, can also save notebooks to **Github** or export

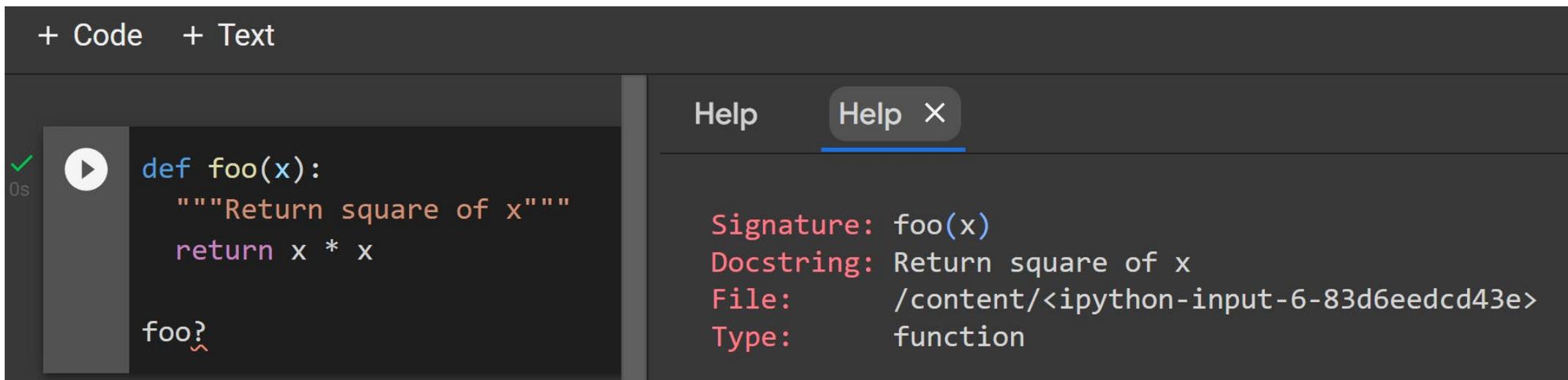
# IPython (getting information)

help(<item>)

<item>?



A screenshot of the IPython interface. On the left, a code cell contains a play button, a checkmark, and the text 'len?'. On the right, a 'Help' panel is open, displaying the signature 'len(obj)', the docstring 'Return the number of items in a container.', and the type 'builtin\_function\_or\_method'.



A screenshot of the IPython interface. At the top, there are buttons for '+ Code' and '+ Text'. Below, a code cell contains a play button, a checkmark, and the code for a function 'foo(x)' that returns the square of x. Below the code is 'foo?'. On the right, a 'Help' panel is open, displaying the signature 'foo(x)', the docstring 'Return square of x', the file path, and the type 'function'.

# IPython (getting source code)

item??

foo??

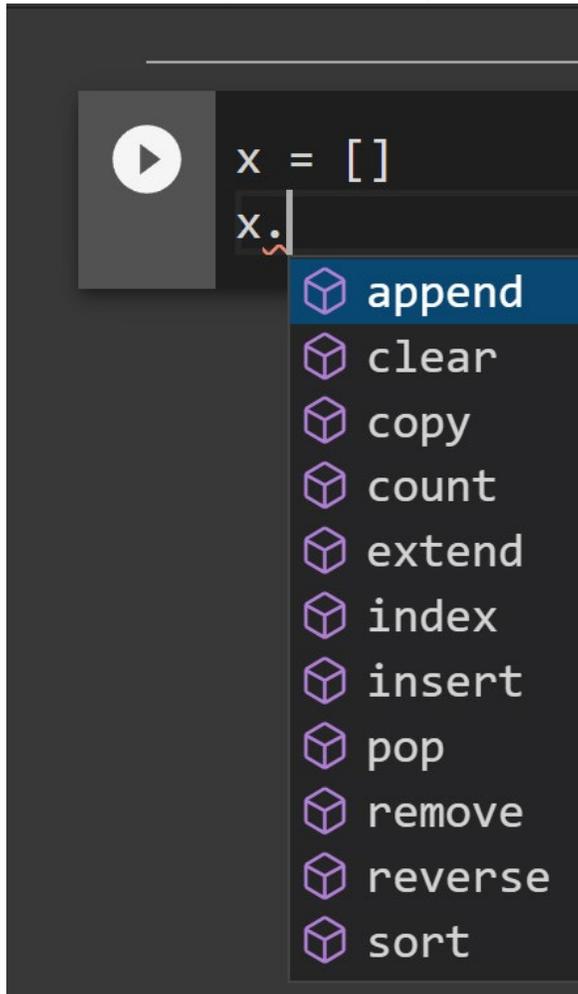
```
0s ✓ ▶ def foo(x):  
    """Return square of x"""  
    return x * x  
  
foo??
```

Help ×

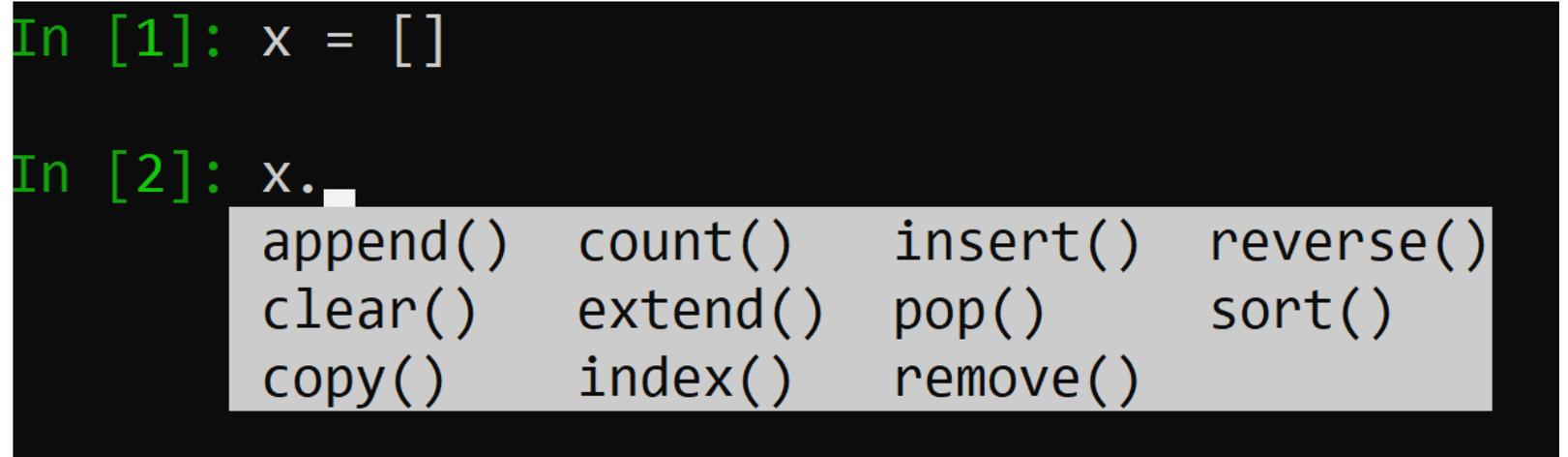
```
Signature: foo(x)  
Source:  
def foo(x):  
    """Return square of x"""  
    return x * x  
File:      /content/<ipython-input-8-a2fe98673a08>  
Type:     function
```

# IPython (getting suggestions – table completion)

Tab completion (mimics the pop-up you get in most IDEs for completion)



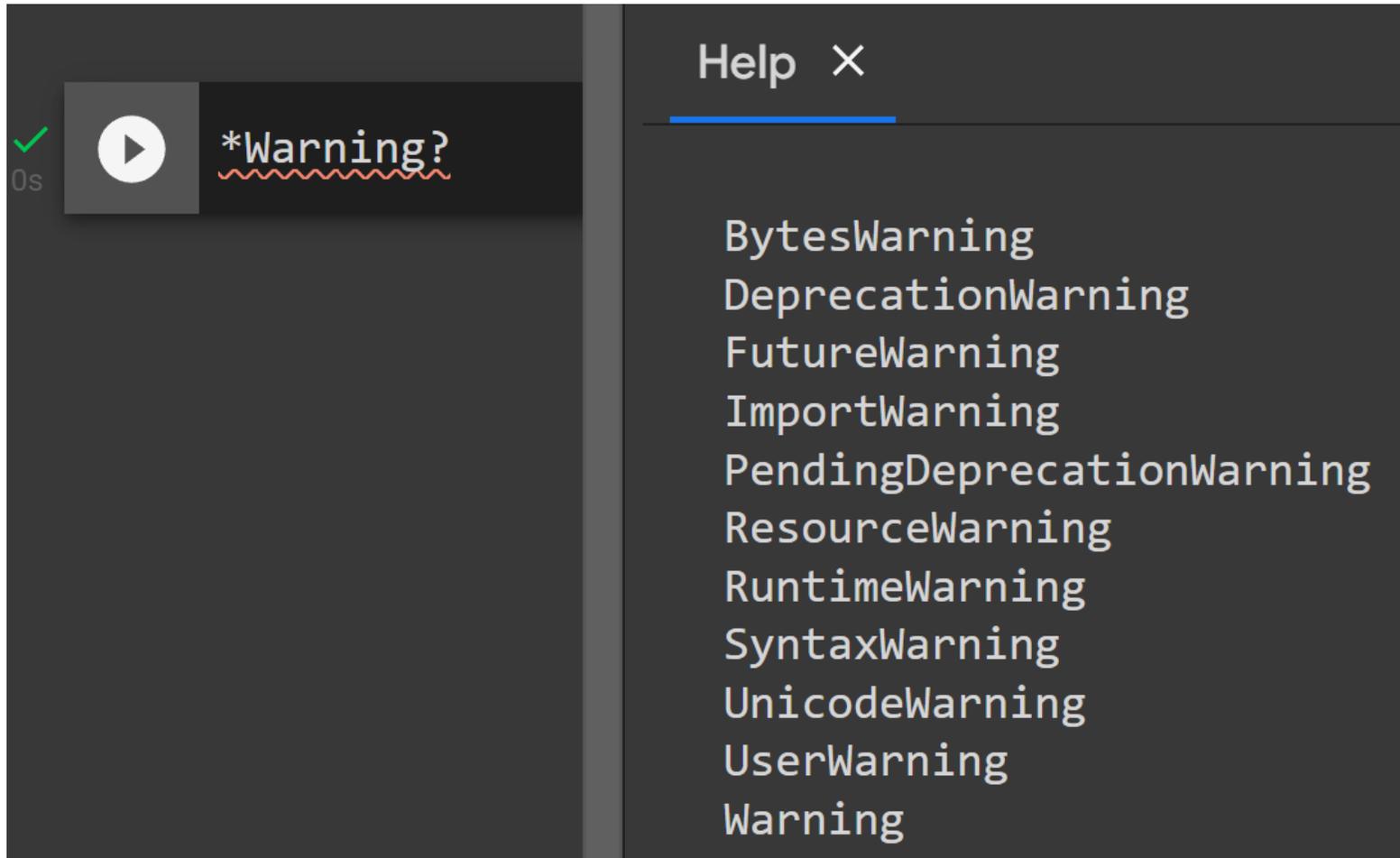
A screenshot of the IPython interface showing a code cell with the text `x = []` and `x.` followed by a cursor. A dropdown menu is open, listing various list methods: `append`, `clear`, `copy`, `count`, `extend`, `index`, `insert`, `pop`, `remove`, `reverse`, and `sort`. Each method name is preceded by a small purple cube icon. The `append` method is highlighted in blue.



A screenshot of the IPython interface showing a code cell with the text `In [1]: x = []` and `In [2]: x.` followed by a cursor. A table completion popup is displayed, showing a grid of list methods: `append()`, `clear()`, `copy()`, `count()`, `extend()`, `index()`, `insert()`, `pop()`, `remove()`, `reverse()`, and `sort()`.

# IPython (getting suggestions – wildcard)

\*Text?

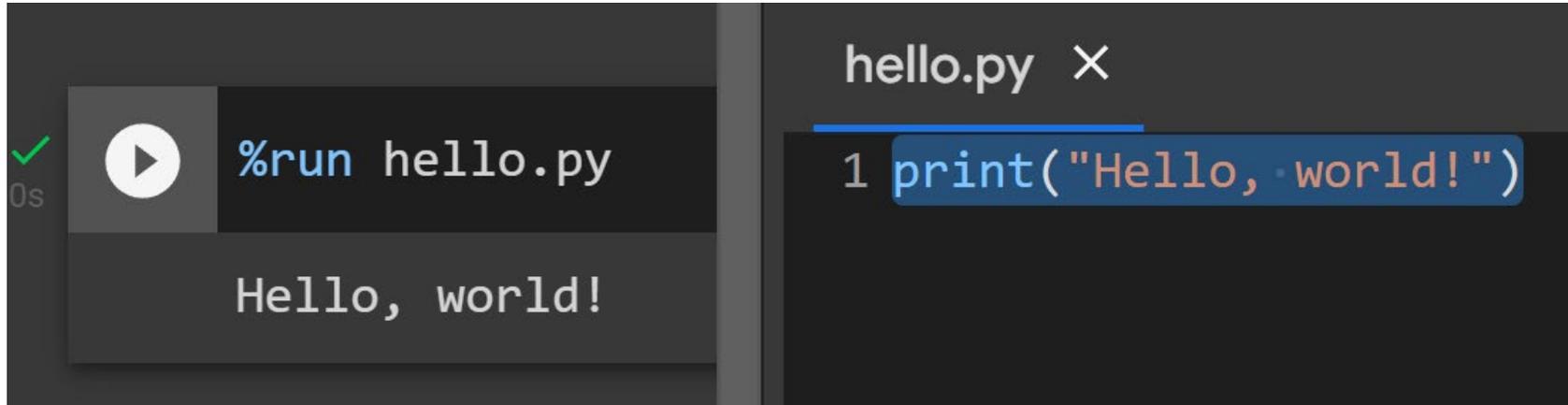


The screenshot shows the IPython interface. On the left, a terminal window displays a green checkmark, '0s', a play button icon, and the text '\*Warning?' with a red wavy underline. On the right, a 'Help X' window is open, listing the following suggestions:

- BytesWarning
- DeprecationWarning
- FutureWarning
- ImportWarning
- PendingDeprecationWarning
- ResourceWarning
- RuntimeWarning
- SyntaxWarning
- UnicodeWarning
- UserWarning
- Warning

# IPython (running external python .py files)

---

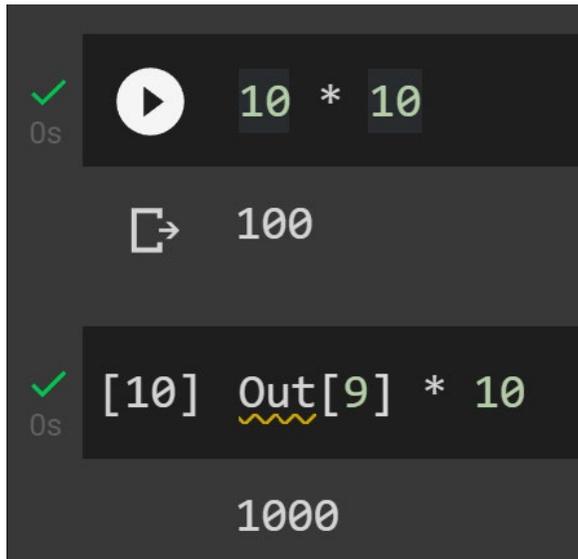


The image shows a screenshot of an IPython environment. On the left, a terminal window displays the command `%run hello.py` being executed, with a green checkmark and a play button icon. The output of the command is `Hello, world!`. On the right, a code editor window titled `hello.py` shows the code `1 print("Hello, world!")` with the line number `1` highlighted in blue.

# IPython (In/Out)

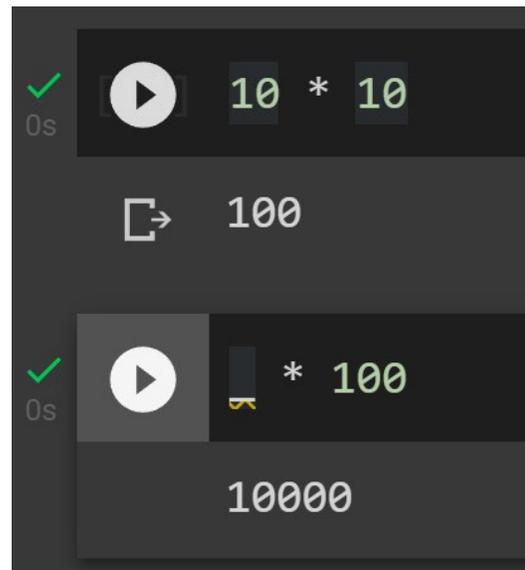
History stored in array blocks **In** and **Out**

Here **In[9]** produced **Out[9] = 100**, so we used it in **In[10]** as input to make **Out[10]**



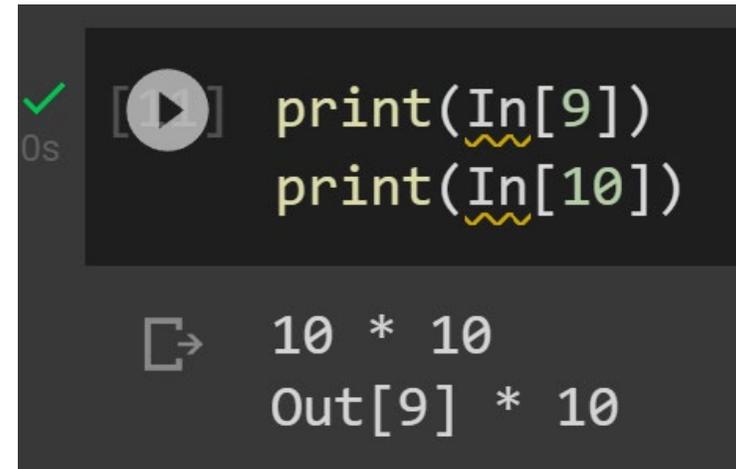
```
✓ 0s [▶] 10 * 10
  [→] 100

✓ 0s [10] Out[9] * 10
      1000
```



```
✓ 0s [▶] 10 * 10
  [→] 100

✓ 0s [▶] 100 * 100
      10000
```



```
✓ 0s [▶] print(In[9])
          print(In[10])

  [→] 10 * 10
      Out[9] * 10
```

*Great when you have a long or large calculation you don't want to re-calculate but didn't store into variable*

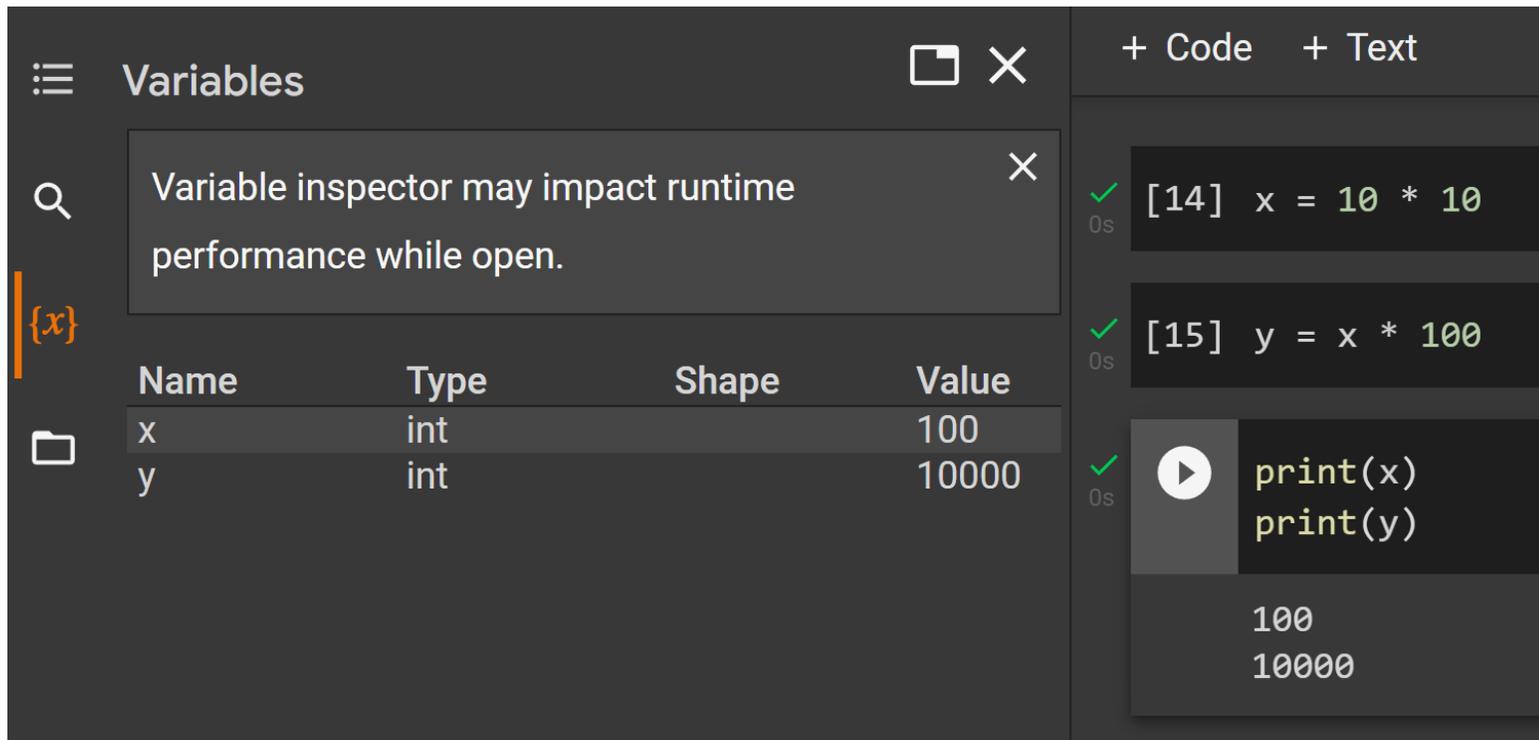
*\_ for previous*

*\_\_ for previous previous*

*\_\_\_ for previous previous previous*

# IPython (variables)

You can also use variables to store data in between blocks which is often a better  
But do remember you have to run a previous block before later can access it



The screenshot shows an IPython notebook interface. On the left, a 'Variables' sidebar displays a table of current variables:

Name	Type	Shape	Value
x	int		100
y	int		10000

The main notebook area contains three code blocks:

```
[14] x = 10 * 10
```

```
[15] y = x * 100
```

```
print(x)  
print(y)
```

The output of the third block is:

```
100  
10000
```

# IPython (terminal commands)

Not only can use access terminal commands using !

But you can also get data from them, or send data out of program

Some commands don't need !, like **run**, cd, mkdir, ls, cp, rm, cat, man, more, mv, ...

```
▶ !echo "hello, world"
!pwd
!ls

↳ hello, world
/content
hello.py sample_data
```

```
▶ x = !pwd
print(x)

↳ ['/content']

[31] message = "Hello, world!"
!echo {message}

Hello, world!
```

# IPython (timing)

---

`%timeit` for one line (`%time` run once)

`%%timeit` for multiple lines (`%%time` run multiple lines once)

```
 %timeit x = [n ** 2 for n in range(1000)]  
 1000 loops, best of 5: 264 µs per loop
```

```
In [2]: %timeit x = [n ** 2 for n in range(1000)]  
230 µs ± 9.36 µs per loop (mean ± std. dev. of 7 runs,  
1,000 loops each)
```

# IPython (timing dangers)

Be wary of stored results in **timeit** (could use time to only run once, or make sure we are sorting random each time)

```
import random
L = [random.random() for i in range(100000)]
%timeit L.sort()
```

The slowest run took 30.01 times longer than the fastest. This could mean that an intermediate result is being cached.  
1000 loops, best of 5: 691 µs per loop

You can also profile something using **prun**

```
import random
L = [random.random() for i in range(100000)]
%prun %timeit L.sort()
```

7412 function calls (7312 primitive calls) in 4.308 seconds

Ordered by: internal time

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
6111	4.269	0.001	4.269	0.001	{method 'sort' of 'list' objects}
9	0.036	0.004	4.305	0.478	<magic-timeit>:1(inner)
5	0.002	0.000	0.002	0.000	socket.py:543(send)
12	0.000	0.000	0.000	0.000	{built-in method builtins.compile}
1	0.000	0.000	4.308	4.308	execution.py:909(timeit)
9	0.000	0.000	4.305	0.478	execution.py:125(timeit)
27/1	0.000	0.000	0.000	0.000	ast.py:320(generic_visit)
71	0.000	0.000	0.000	0.000	ast.py:193(iter_child_nodes)
138	0.000	0.000	0.000	0.000	{built-in method builtinsgetattr}
1	0.000	0.000	3.472	3.472	timeit.py:183(repeat)
36/1	0.000	0.000	0.000	0.000	ast.py:153(fix)

*I profiled the **timeit** command*

*It ran 6111 samples*

# Onward to ... coding!

---

Jonathan Hudson  
[jwhudson@ucalgary.ca](mailto:jwhudson@ucalgary.ca)  
<https://pages.cpsc.ucalgary.ca/~hudsonj/>

