

Machine Learning: Optional: IPython

CPSC 501: Advanced Programming Techniques Winter 2025

Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Thursday, February 13, 2025

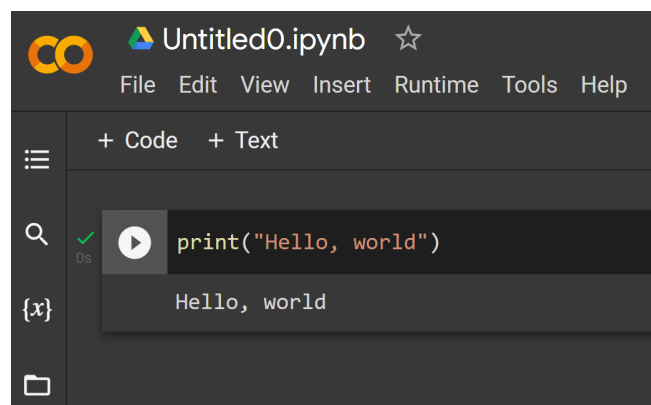
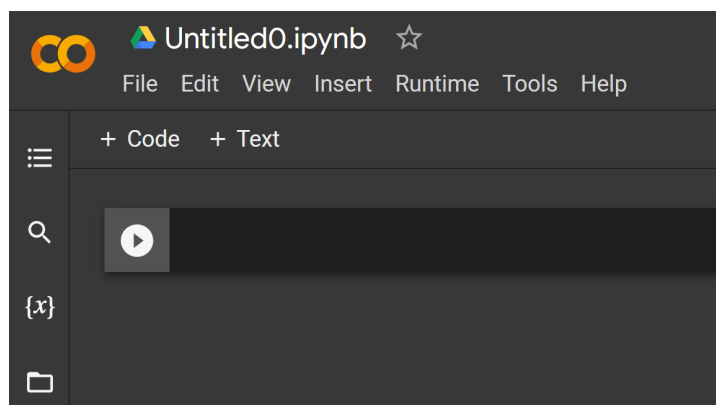
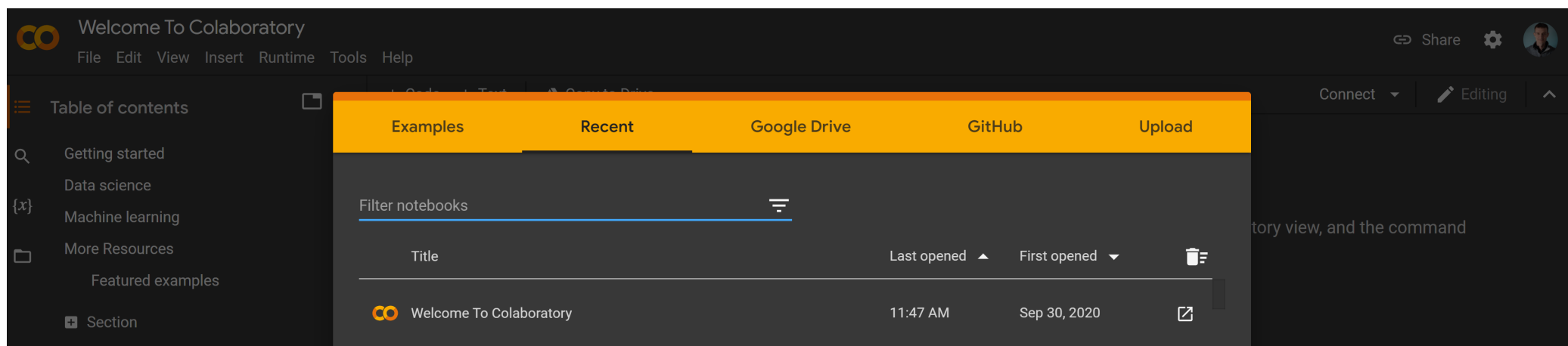
Copyright © 2025



UNIVERSITY OF
CALGARY

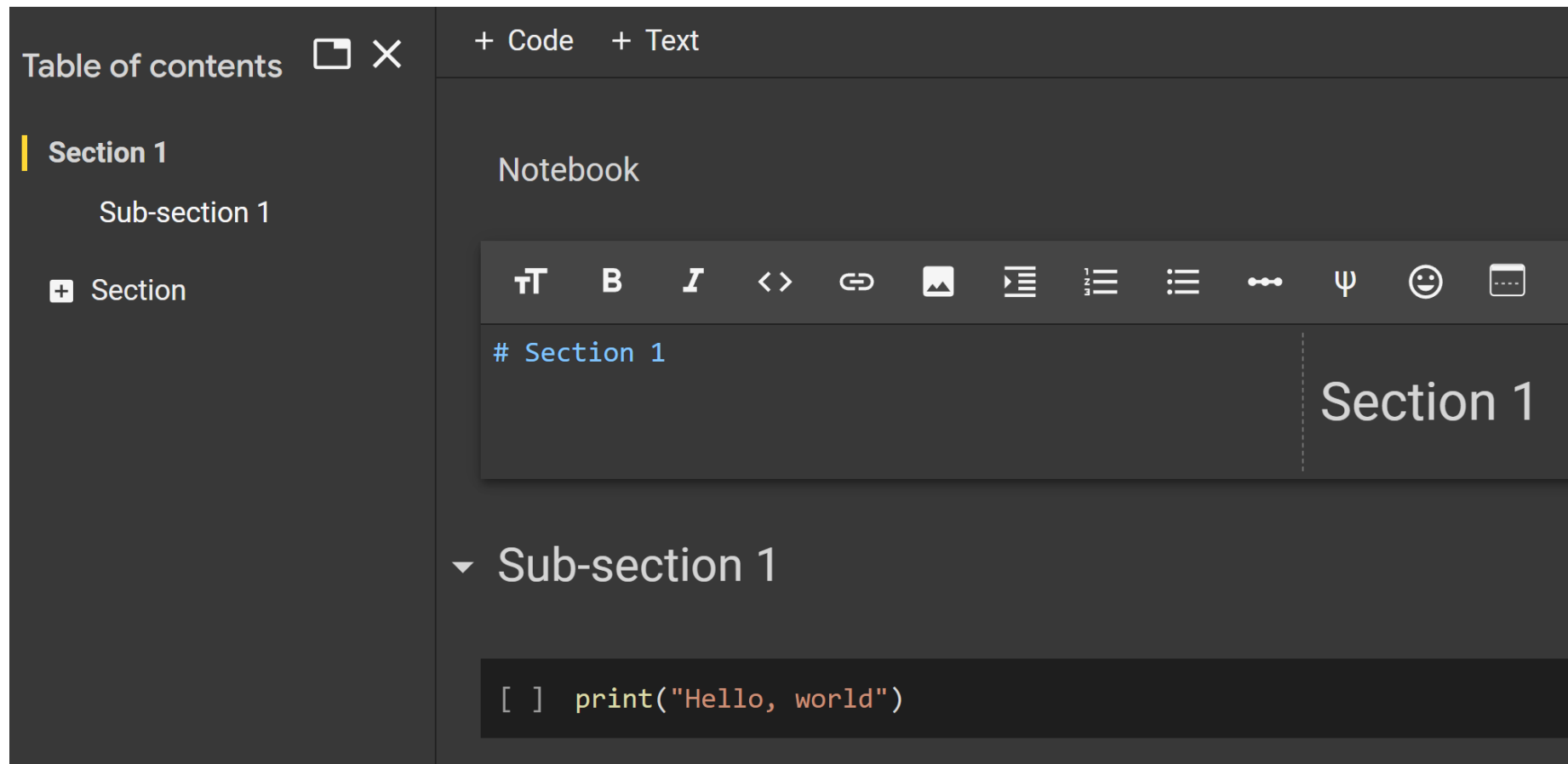
Beginner?

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



IPython (markdown language)

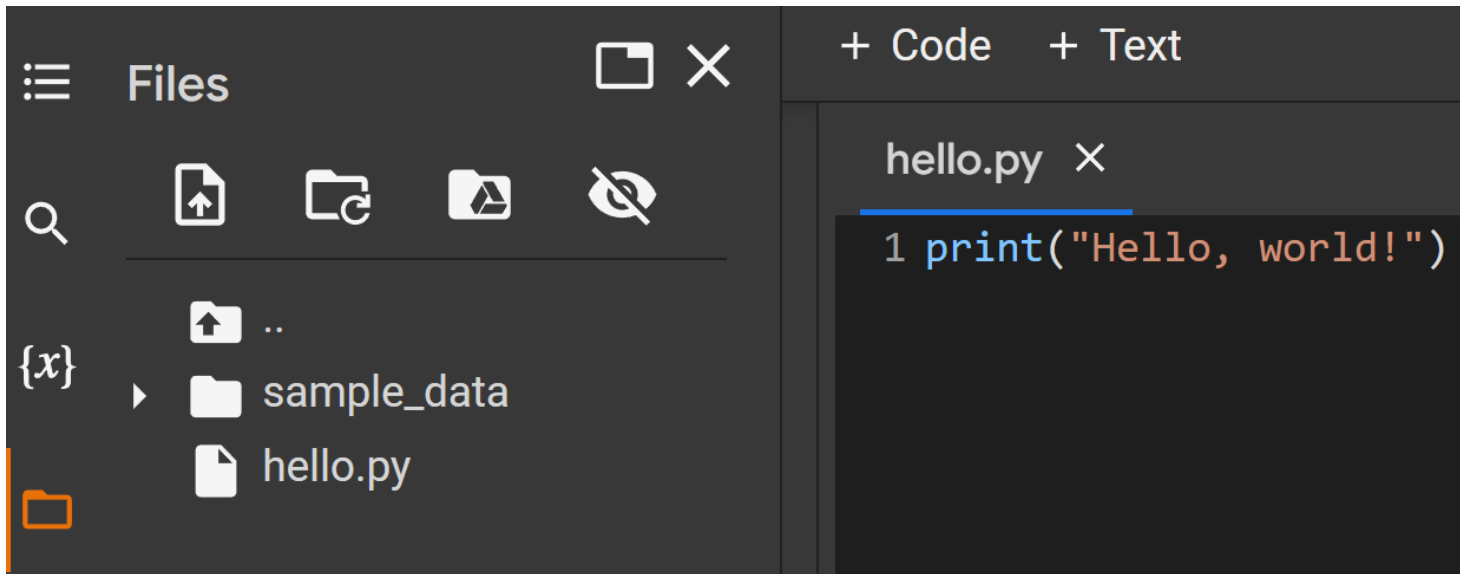
<https://www.markdownguide.org/cheat-sheet/>



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 contains a code cell with the markdown text '# Section 1'. To the right of this cell, the text 'Section 1' is displayed. Below the code cell is a 'Sub-section 1' header. At the bottom, a code cell contains the Python code `[] print("Hello, world")`. The interface includes a top bar with '+ Code' and '+ Text' buttons, and a rich text toolbar with icons for bold, italic, code, link, image, list, table, link, math, emoji, and more options.

IPython (files)

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

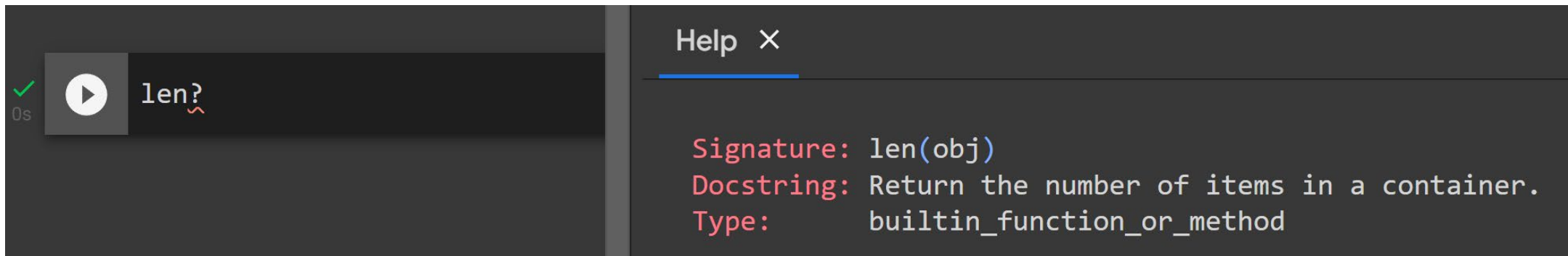


- By default saved in **Google Drive**, can also save notebooks to **Github** or export

IPython (getting information)

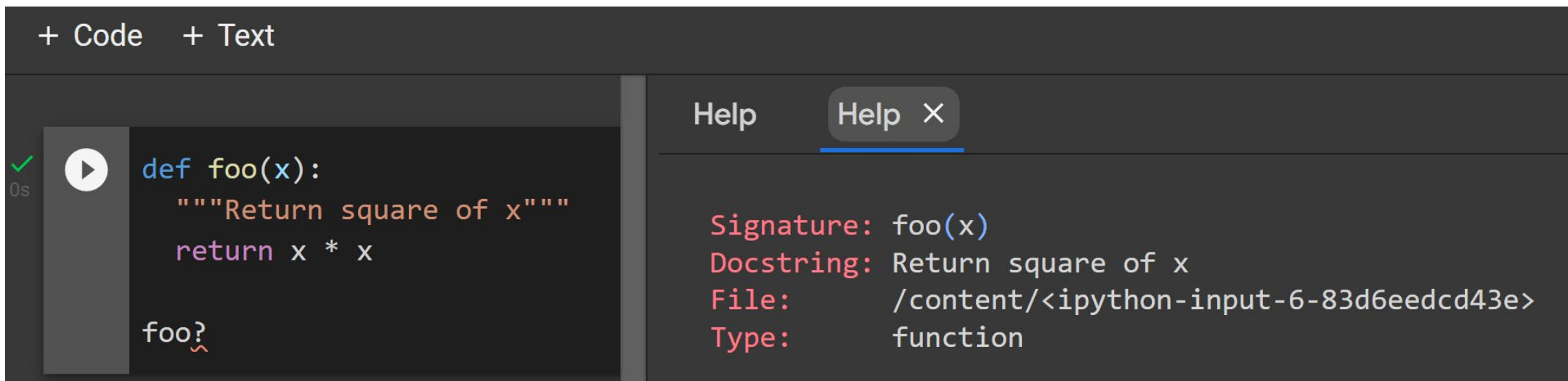
help(<item>)

<item>?



The image shows the IPython interface with a code cell on the left and a help panel on the right. The code cell contains a play button, a green checkmark, and the text 'len?'. The help panel has a title 'Help' with a close button 'X' and displays the following information:

```
Signature: len(obj)
Docstring: Return the number of items in a container.
Type:      builtin_function_or_method
```



The image shows the IPython interface with a code cell on the left and a help panel on the right. The code cell contains a play button, a green checkmark, and the following code:

```
def foo(x):
    """Return square of x"""
    return x * x

foo?
```

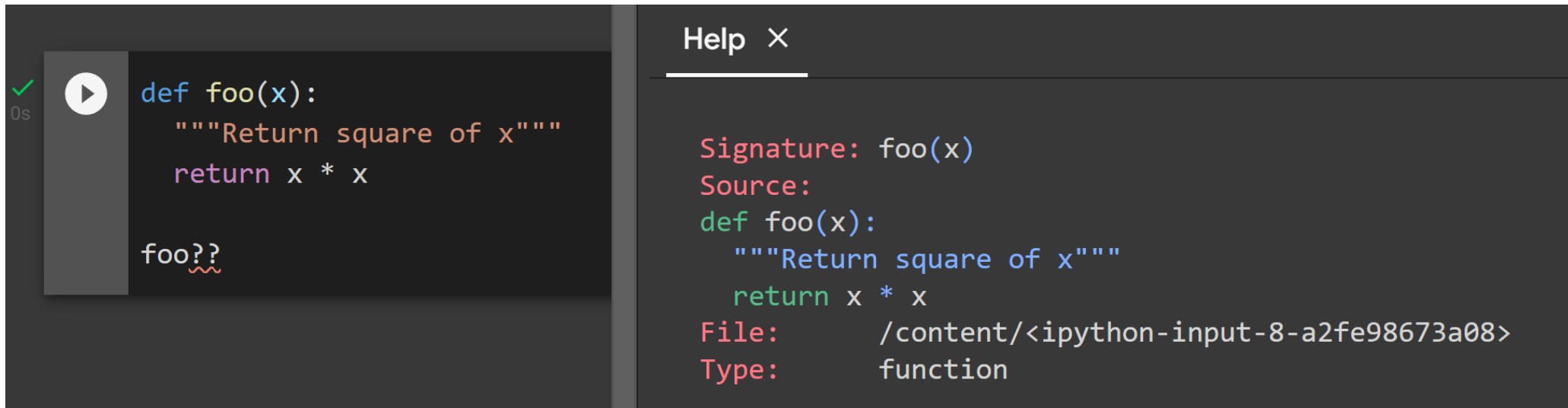
The help panel has a title 'Help' with a close button 'X' and displays the following information:

```
Signature: foo(x)
Docstring: Return square of x
File:      /content/<ipython-input-6-83d6eedcd43e>
Type:      function
```

IPython (getting source code)

item??

foo??



The screenshot shows an IPython environment. On the left, a code cell contains the definition of a function `foo(x)` which returns the square of `x`. Below the code, the text `foo??` is entered, with a red squiggly line under the second question mark. On the right, the 'Help' panel is open, displaying the signature `foo(x)`, the source code of the function, and the file and type information.

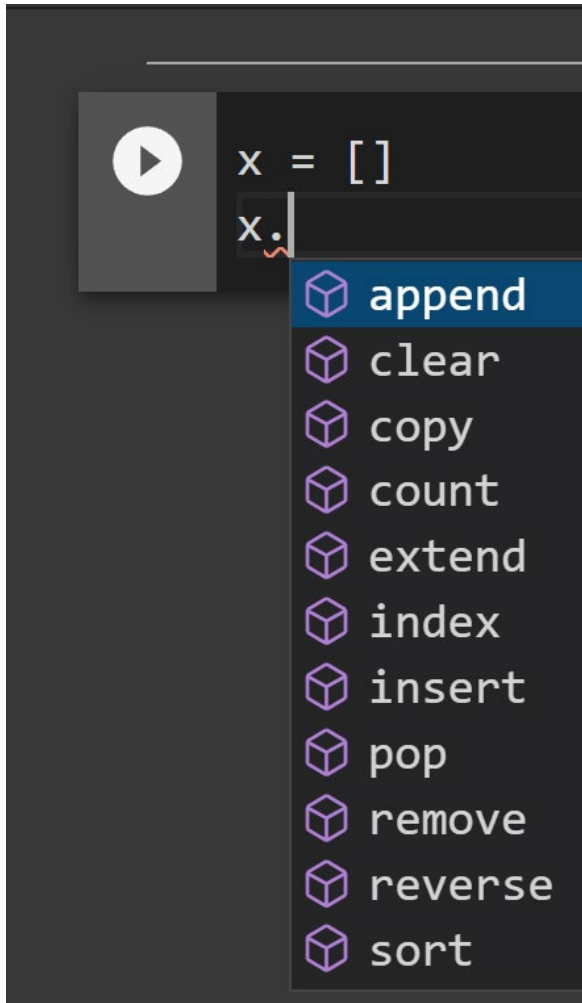
```
def foo(x):  
    """Return square of x"""  
    return x * x  
  
foo??
```

Help X

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)

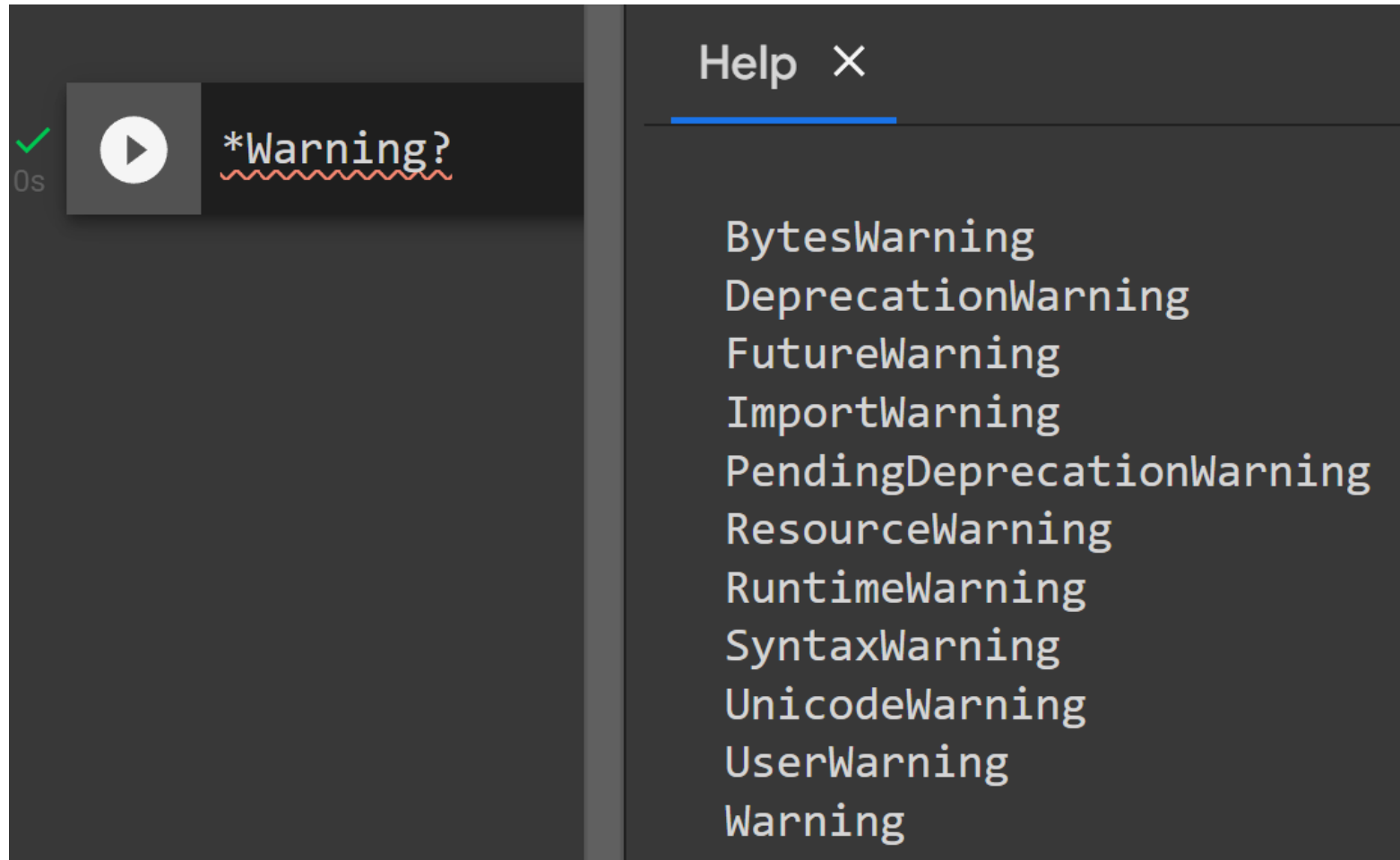
Tab completion (mimics the pop-up you get in most IDEs for completion)



```
In [1]: x = []  
  
In [2]: x.  
append()  count()  insert()  reverse()  
clear()   extend()  pop()     sort()  
copy()    index()   remove()
```

IPython (getting suggestions – wildcard)

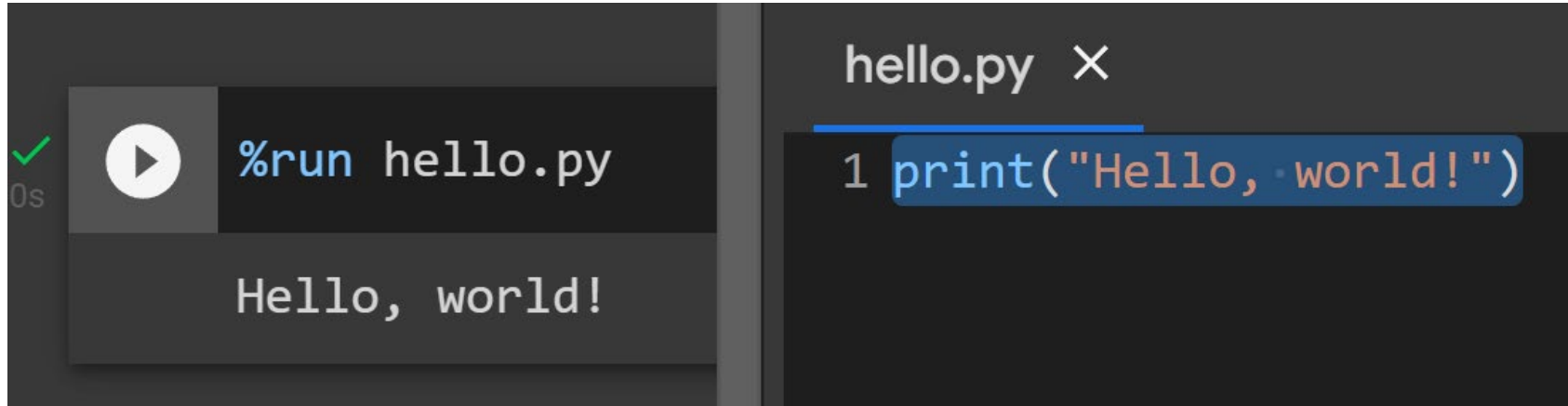
*Text?



The screenshot shows the IPython interactive help interface. On the left, a dark grey sidebar contains a green checkmark, a play button icon, and the text `*Warning?` with a red wavy underline. Below this, the text `0s` is visible. The main area on the right has a dark grey background with a 'Help X' header. A list of suggestions is displayed in a monospaced font:

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

IPython (running external python .py files)



```
hello.py x
1 print("Hello, world!")
```

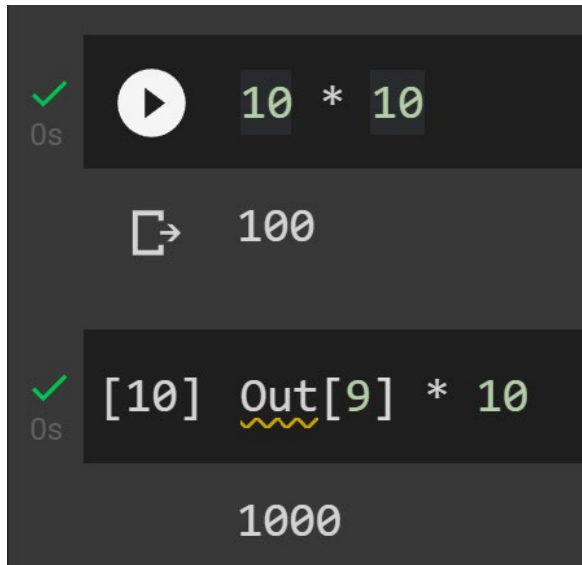
✓ 0s %run hello.py

Hello, world!

IPython (In/Out)

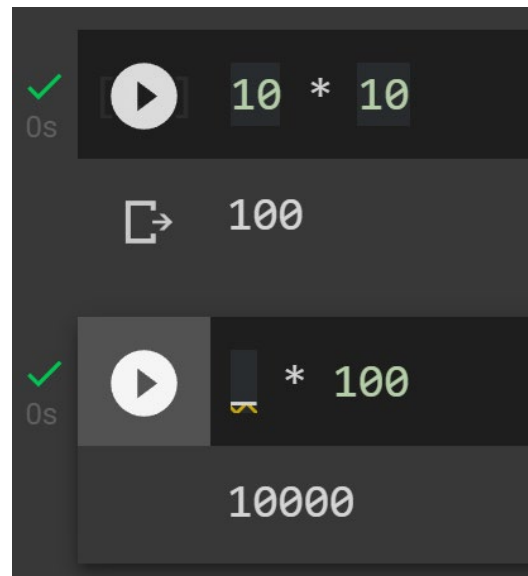
History stored in array blocks **In** and **Out**

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



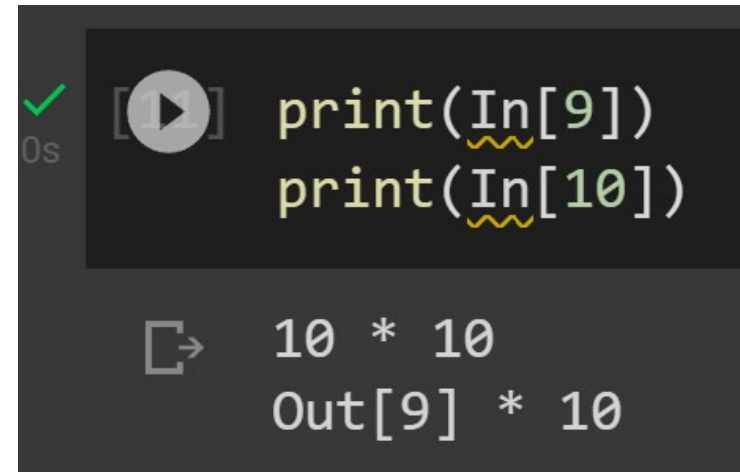
```
✓ 0s [▶] 10 * 10
    [→] 100

✓ 0s [10] Out[9] * 10
    1000
```



```
✓ 0s [▶] 10 * 10
    [→] 100

✓ 0s [▶] [10] * 100
    10000
```



```
✓ 0s [▶] print(In[9])
    print(In[10])
    [→] 100
    1000

    [→] 10 * 10
    Out[9] * 10
    100
    1000
```

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

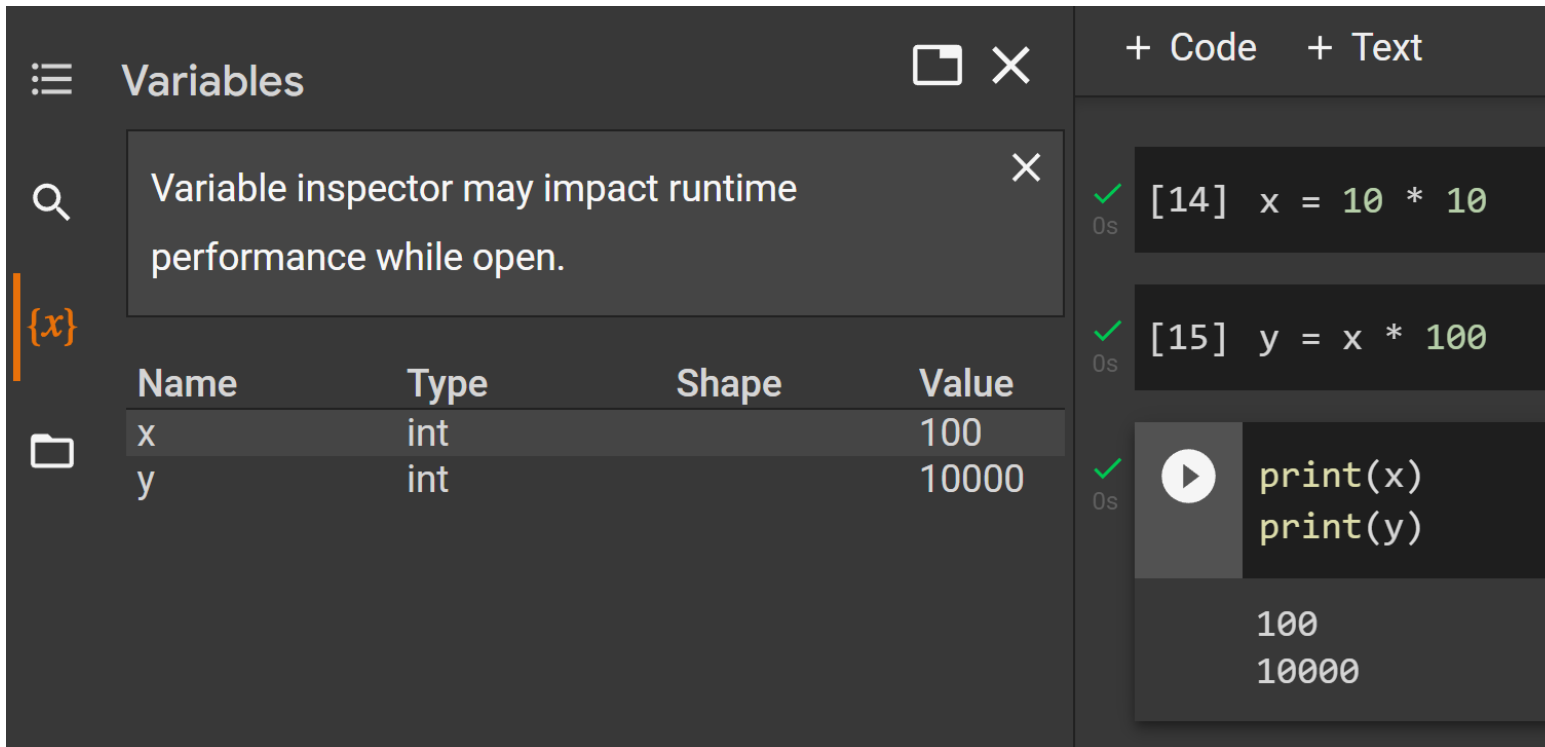
_ for previous

__ 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 displays the IPython interface. On the left, the 'Variables' panel is open, showing a table of current variables. A warning message is also visible: 'Variable inspector may impact runtime performance while open.' The table lists two variables: 'x' with type 'int' and value '100', and 'y' with type 'int' and value '10000'. On the right, the code editor shows three code blocks. The first block contains 'x = 10 * 10', the second contains 'y = x * 100', and the third contains 'print(x)' and 'print(y)'. The output of the third block is displayed below it, showing '100' and '10000'.

Name	Type	Shape	Value
x	int		100
y	int		10000

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

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

```
print(x)
```

```
print(y)
```

```
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']
```

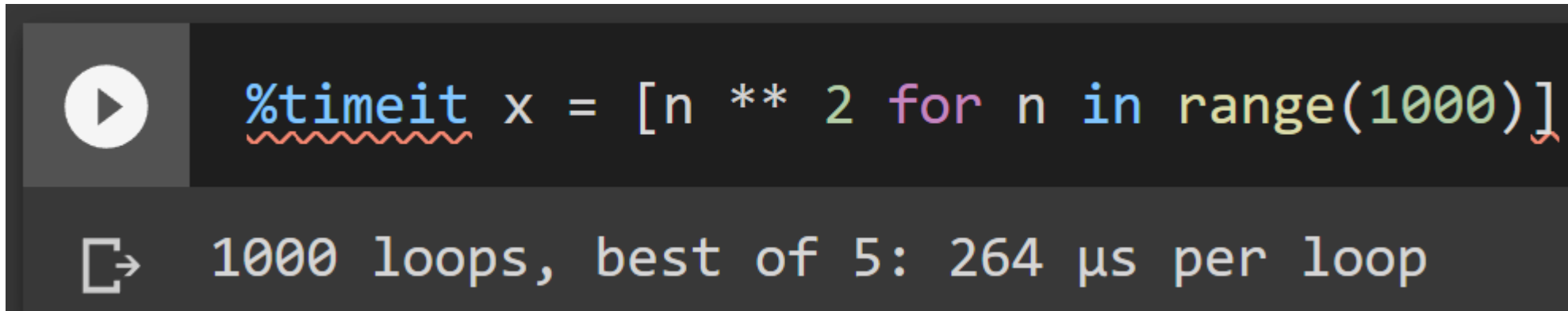
```
[31] message = "Hello, world!"  
!echo {message}
```


```
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()
```

*I profiled the **timeit** command*

It ran 6111 samples

7412 function calls (7312 primitive calls) in 4.308 seconds

Ordered by: internal time

ncalls	totttime	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)

Onward to ... Optional Advanced Python

Jonathan Hudson
jwhudson@ucalgary.ca
<https://pages.cpsc.ucalgary.ca/~jwhudson/>



UNIVERSITY OF
CALGARY