

Data Science: IPython

CPSC 501: Advanced Programming Techniques
Fall 2022

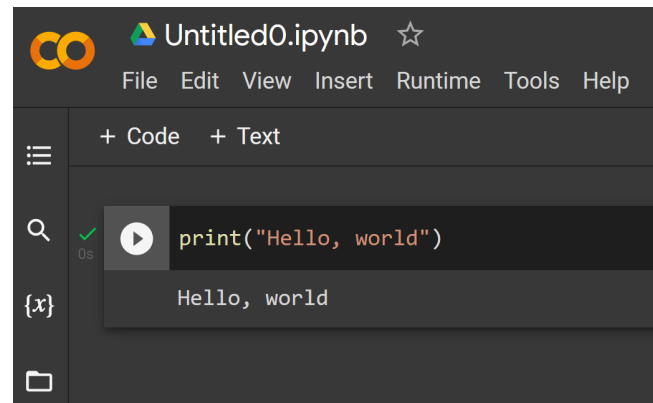
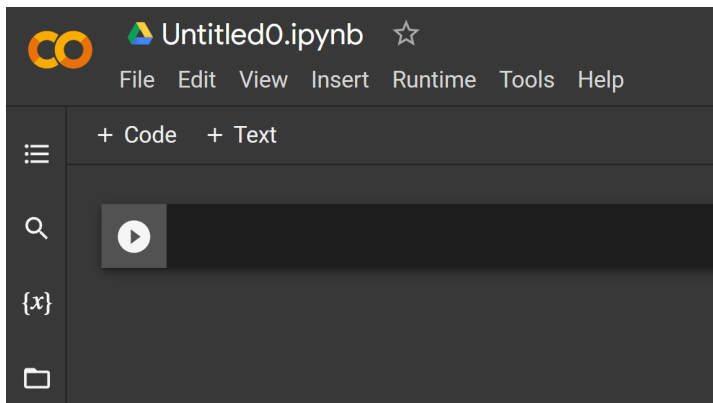
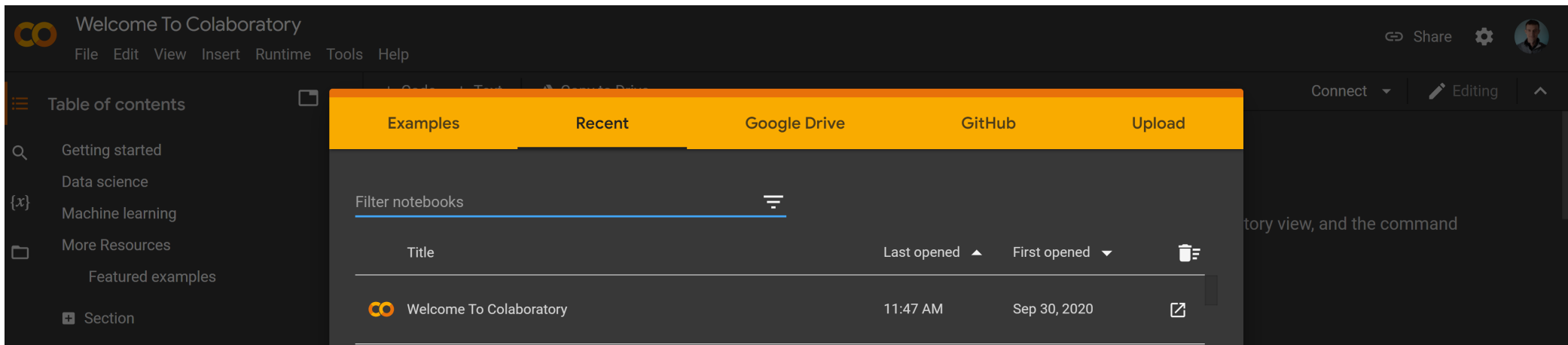
Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Monday, October 3, 2022



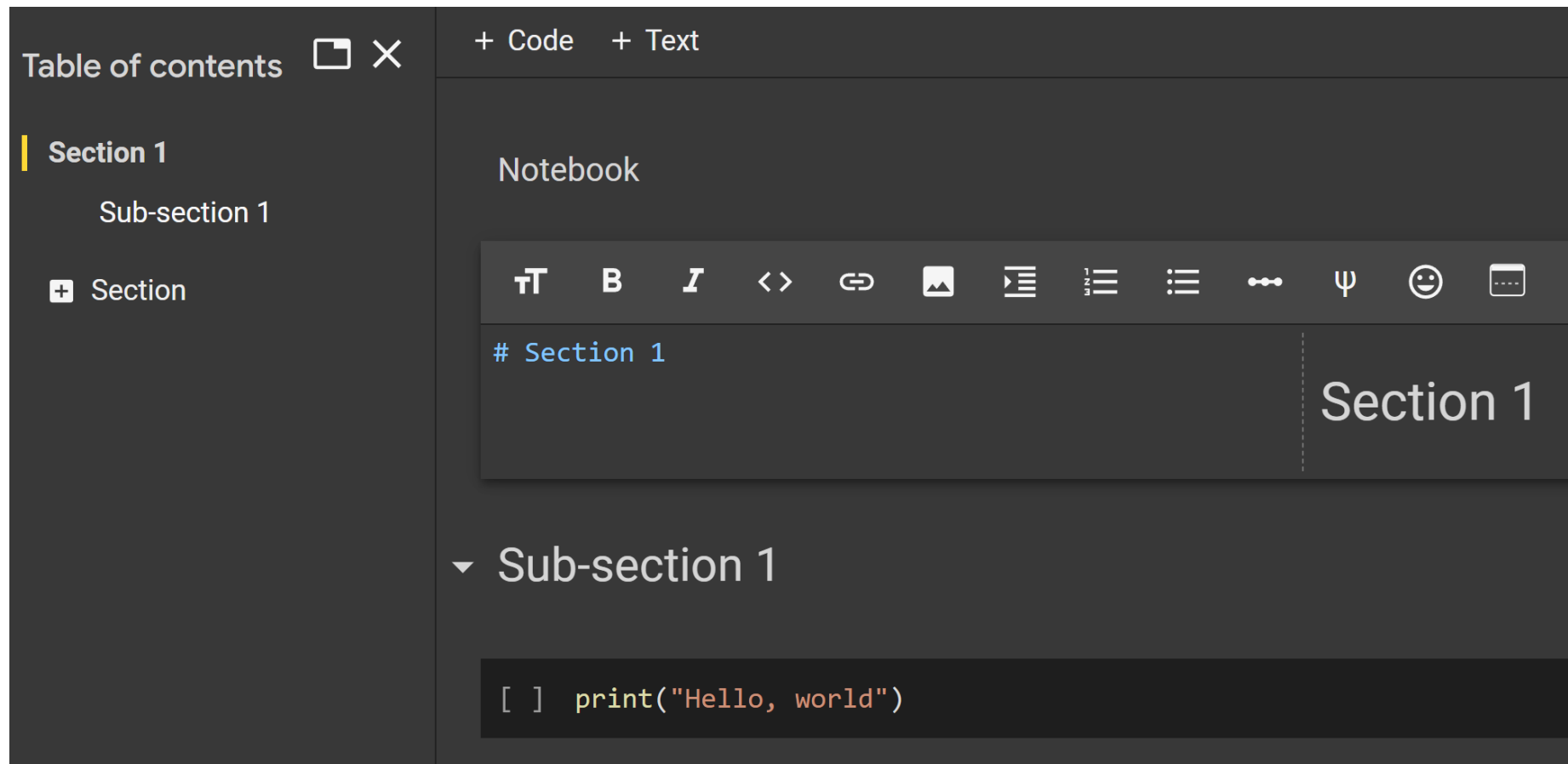
Beginner?

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



IPython (markdown language)

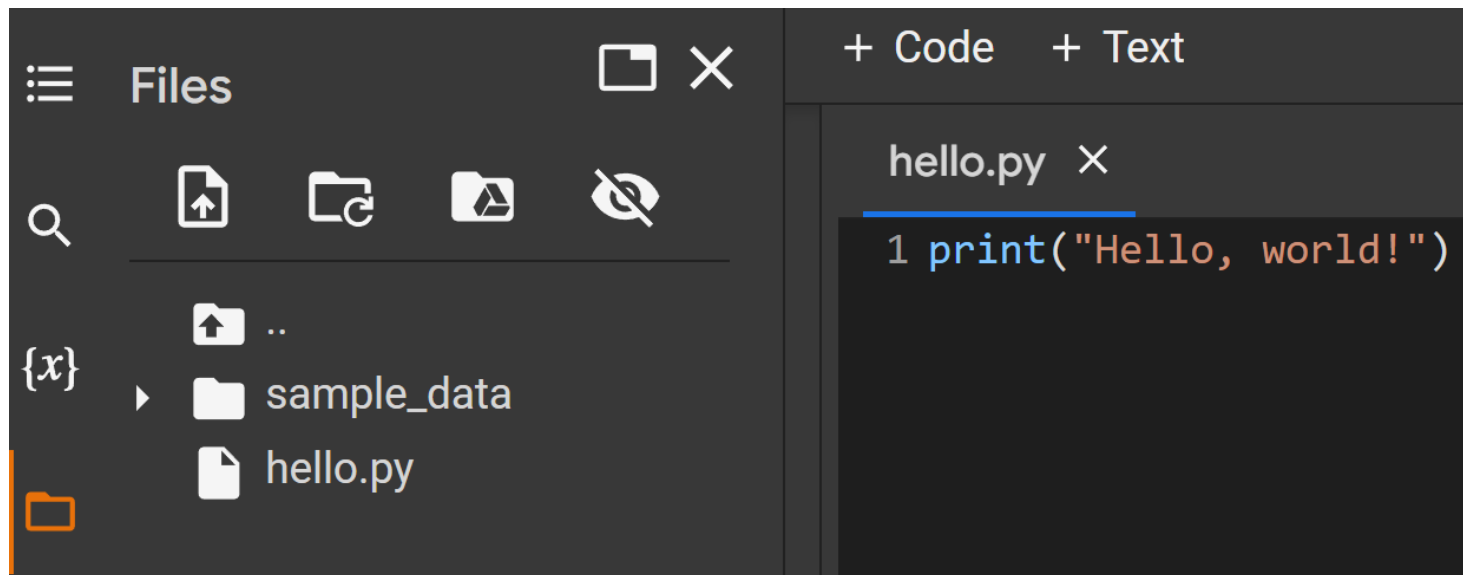
<https://www.markdownguide.org/cheat-sheet/>



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, there are two cells: a markdown cell containing '# Section 1' which has rendered as 'Section 1' on the right, and a code cell containing the Python code `[] print("Hello, world")`.

IPython (files)

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

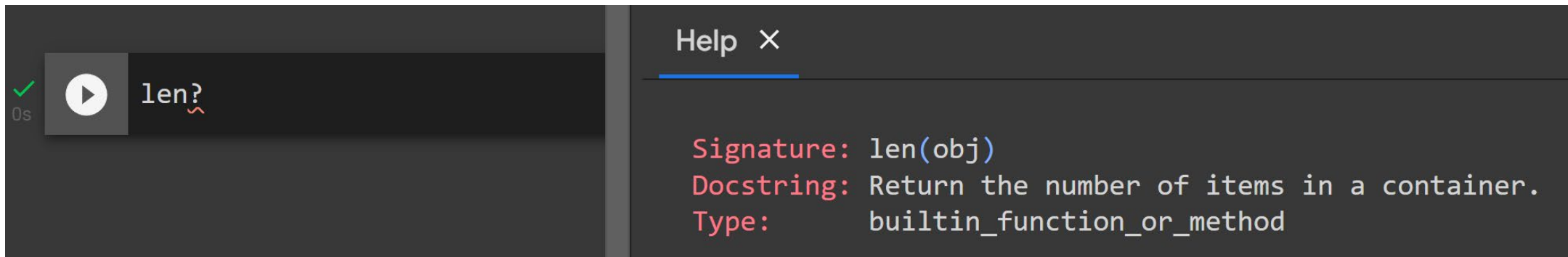


- By default saved in **Google Drive**, can also save notebooks to **Github** or export

IPython (getting information)

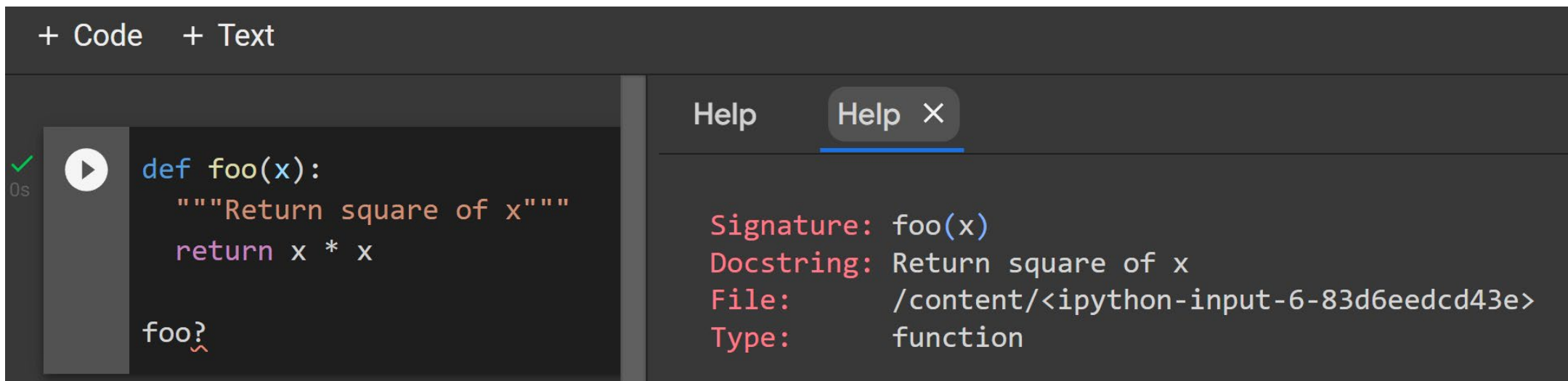
help(<item>)

<item>?



A screenshot of the IPython interface. On the left, a code cell contains a play button, a green checkmark, and the text 'len?'. On the right, a 'Help' panel is open, displaying the following information:

```
Signature: len(obj)
Docstring: Return the number of items in a container.
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 green checkmark, and the following Python code:

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

Below the code, the text 'foo?' is entered. On the right, a 'Help' panel is open, displaying 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??

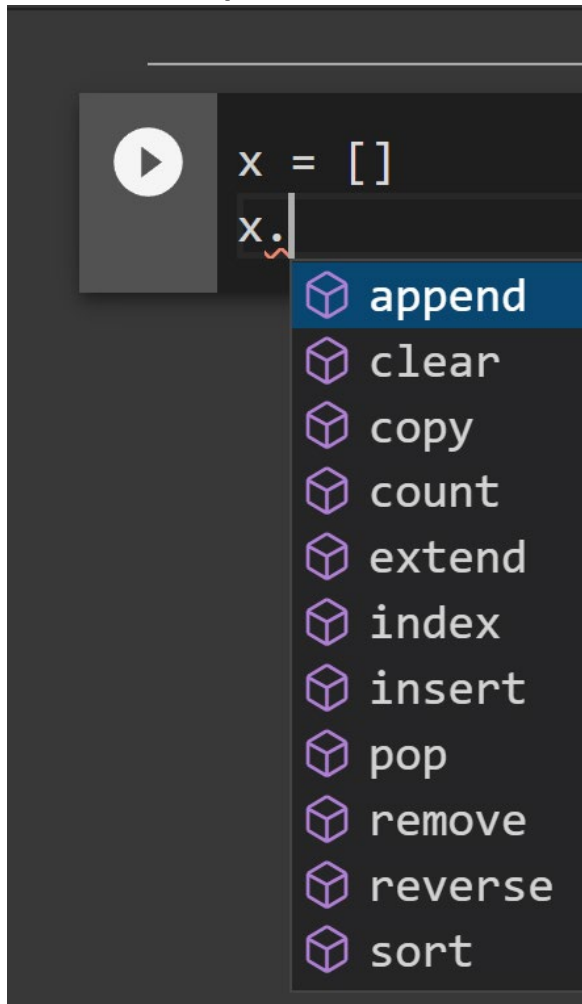
```
0s ✓ [play] def foo(x):  
    """Return square of x"""  
    return x * x  
  
foo??
```

Help ×

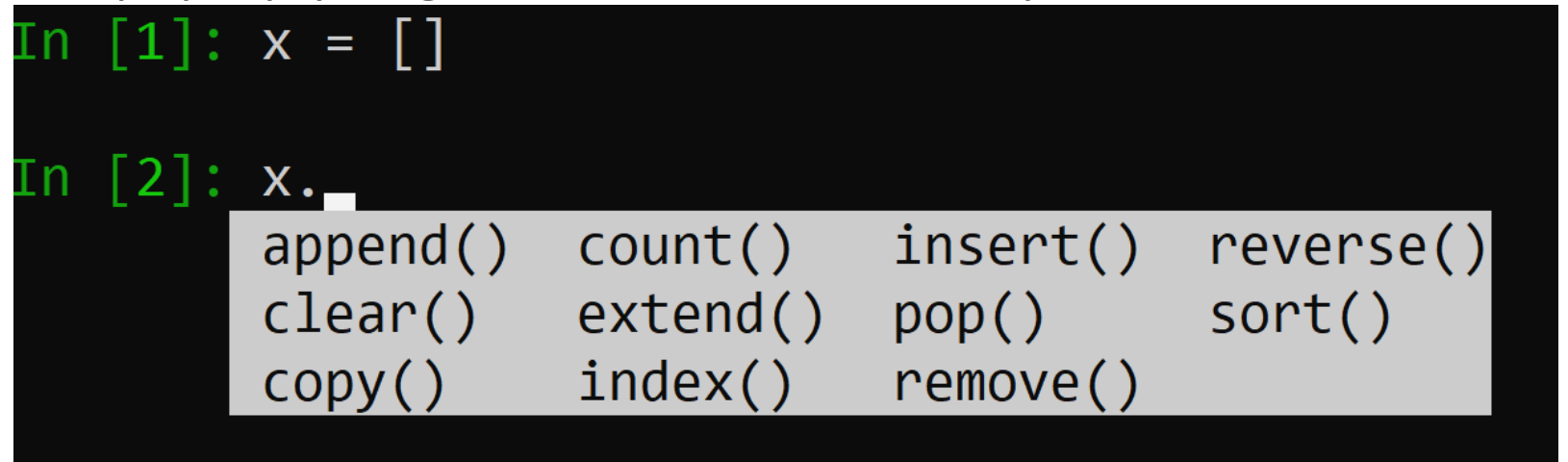
```
Signature: foo(x)  
Source:  
def foo(x):  
    """Return square of x"""  
    return x * x  
File:      /content/<ipython-input-8-a2fe98673a08>  
Type:     function
```

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.` on the next line. A dropdown menu is open below `x.`, 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 code completion popup. The background is black with green text for the prompt `In [1]: x = []` and `In [2]: x.`. A light gray popup box displays a grid of list methods: `append()`, `clear()`, `copy()`, `count()`, `extend()`, `index()`, `insert()`, `pop()`, `remove()`, `reverse()`, and `sort()`.

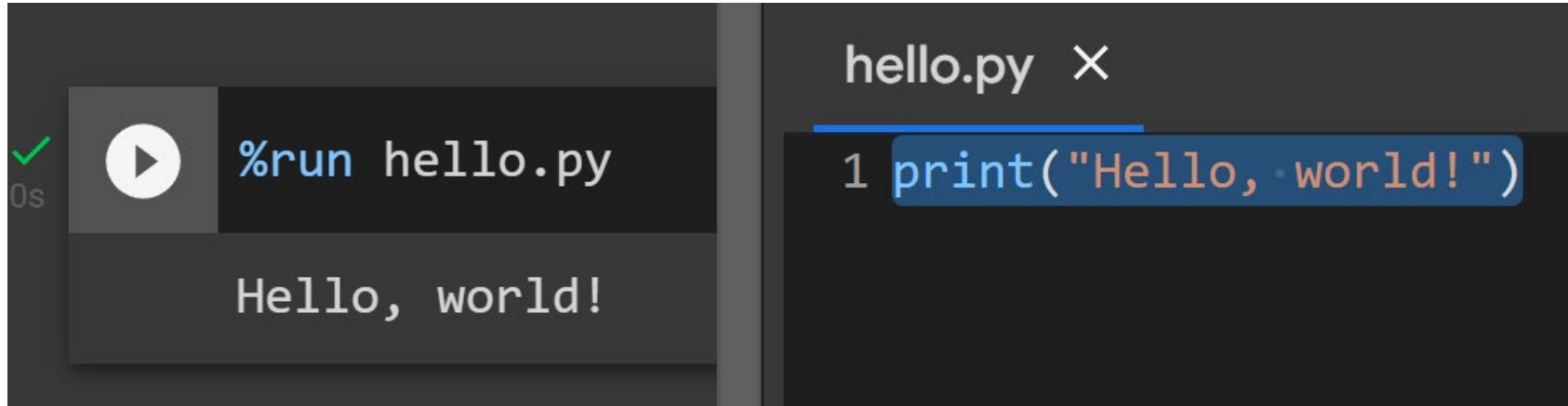
IPython (getting suggestions – wildcard)

*Text?

The screenshot shows the IPython help window. 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)

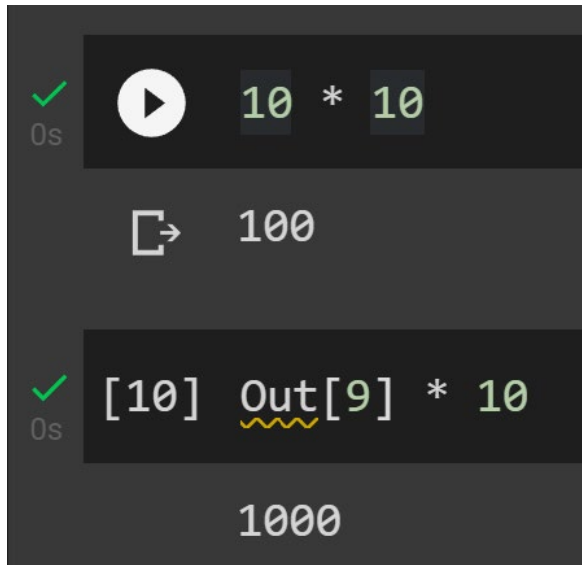


The image shows a screenshot of an IPython environment. On the left, a terminal window displays the command `%run hello.py` being executed, with a green checkmark and a play button icon. The output of the command is `Hello, world!`. On the right, a code editor window titled `hello.py` shows the code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

IPython (In/Out)

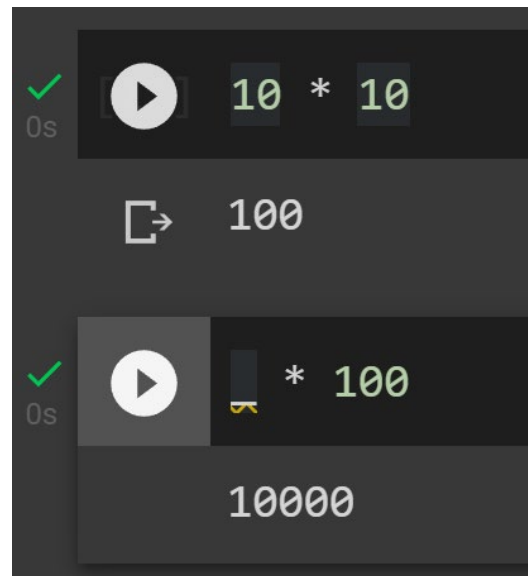
History stored in array blocks **In** and **Out**

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



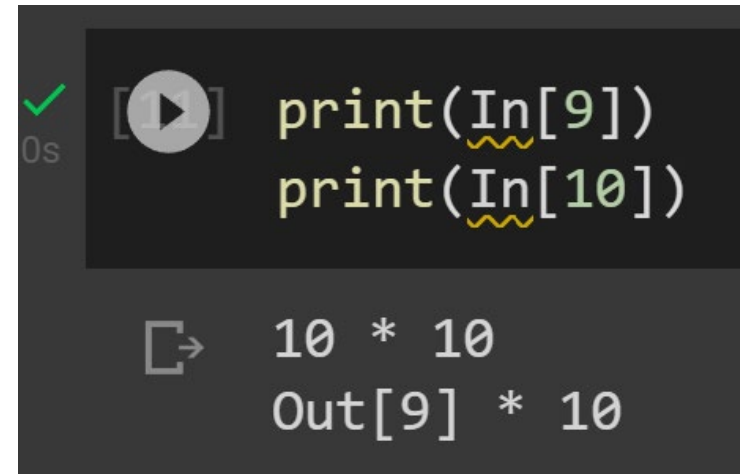
```
✓ 0s [▶] 10 * 10
  [→] 100

✓ 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

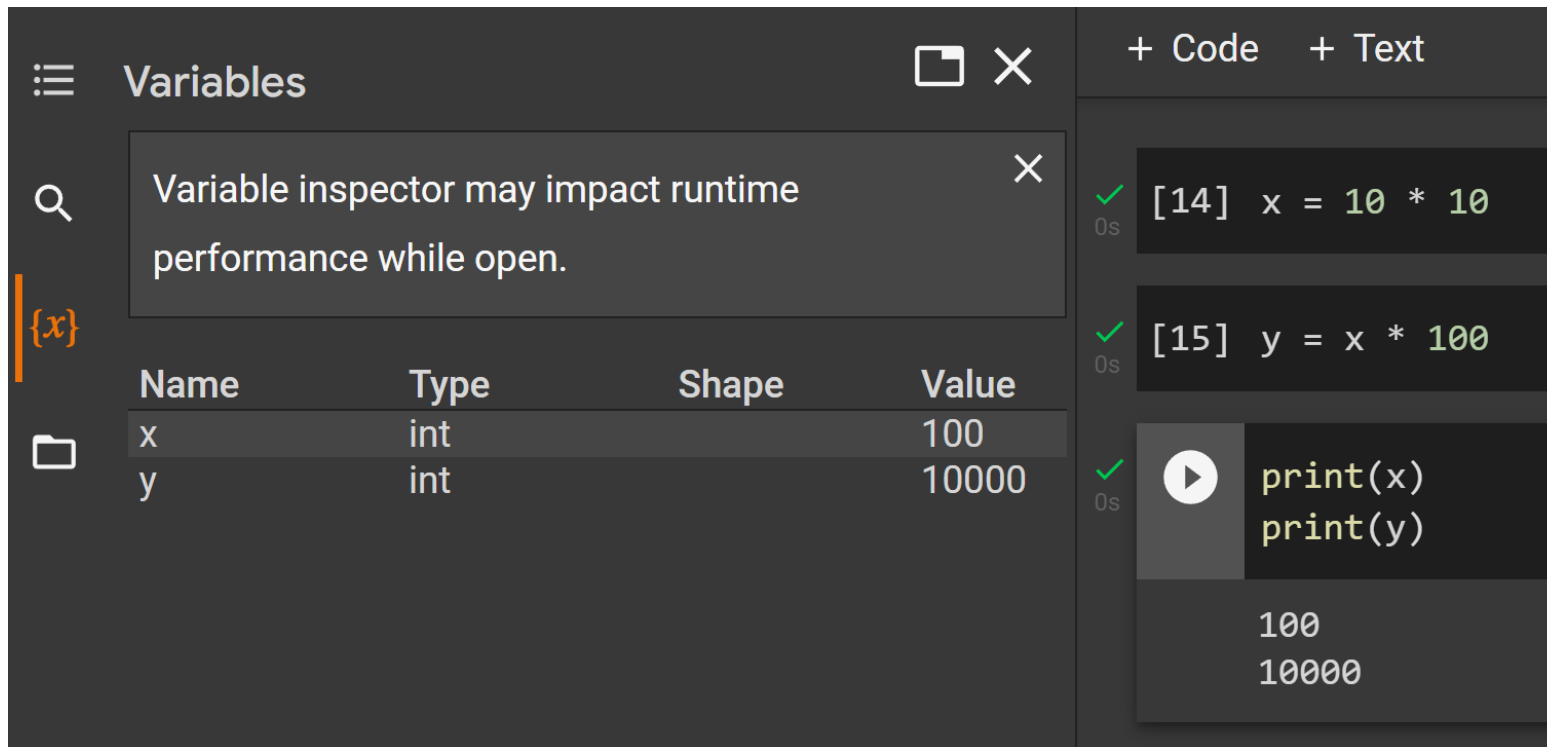
_ for previous

__ 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 a Jupyter Notebook interface. On the left, the 'Variables' panel is open, displaying a table of variables:

Name	Type	Shape	Value
x	int		100
y	int		10000

On the right, the code editor shows 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!
```

Some More Common Advanced Python Seen in Data Science

Generators

- Function that returns on each yield call, tracking state so it can be restarted to continue

```
def squares(n):  
    for i in range(n + 1):  
        yield i * i  
  
for i in squares(5):  
    print(i)
```

```
0  
1  
4  
9  
16  
25
```

```
def frange(start, stop, step):  
    if (step > 0):  
        while(start < stop):  
            yield start  
            start += step  
    else:  
        while(start > stop):  
            yield start  
            start += step
```

Can make a quick float range function this way for looping

```
for i in frange(0,1,+0.25):  
    print(i)  
  
for i in frange(1,0,-0.25):  
    print(i)
```

```
0  
0.25  
0.5  
0.75  
1  
0.75  
0.5  
0.25
```

Enumerate

Creates tuple with incrementing index

Input can be any generator

```
▶ for (i, value) in enumerate("Hello, world!"):
    print(i, value)
```

```
0 H
1 e
2 l
3 l
4 o
5 ,
6
7 w
8 o
9 r
10 l
11 d
12 !
```

```
▶ for (i, value) in enumerate(squares(6)):
    print(i, value)
```

```
0 0
1 1
2 4
3 9
4 16
5 25
6 36
```

Zip

- Put two similar length things together (list/tuples)
- Extra length part in either list would be ignored



```
i = ["Toronto", "Calgary"]  
j = ["Ontario", "Alberta"]  
k = zip(i, j)  
print(list(k))
```



```
[('Toronto', 'Ontario'), ('Calgary', 'Alberta')]
```


Zip to dictionary

- Dict via zip mapping



```
keys = ["Toronto", "Calgary"]  
values = ["Ontario", "Alberta"]  
cities = dict(zip(keys, values))  
print(cities)
```

```
☞ {'Toronto': 'Ontario', 'Calgary': 'Alberta'}
```

Data structure comprehension

- Create list in-line
- dict/set applicable

```
1 [expr for val in collection]
2 [expr for val in collection if condition]
3 {expr for val in collection if condition}
4 {key-expr:val-expr for val in collection if condition}
```

```
▶ temp = [x * x for x in range(10)]
print(temp)
temp = [(x, x * x) for x in range(10)]
print(temp)
temp = [(x, x * x) for x in range(10) if x * x > 50]
print(temp)

temp = {(x, x * x) for x in range(10) if x * x > 50}
print(temp)

temp = {x:x*x for x in range(10) if x * x > 50}
print(temp)

↳ [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81)]
[(8, 64), (9, 81)]
{(8, 64), (9, 81)}
{8: 64, 9: 81}
```

Mapping

- Map a function onto another collection

```
▶ list(map(len, ["This", "is", "hello", "world", "!"]))
```

```
↳ [4, 2, 5, 5, 1]
```

```
[53] set(map(len, ["This", "is", "hello", "world", "!"]))
```

```
{1, 2, 4, 5}
```

Mapping – functions are objects

- Even your own function can be applied as mapping

```
def foo(x):  
    return x*x  
  
print(list(map(foo, range(5))))  
  
[0, 1, 4, 9, 16]
```

Lambdas

- Lambda – make short function into inline expression (easy for mapping)

```
▶ def foo(x):  
    return x*x  
  
lambda x: x*x  
  
print(list(map(foo, range(5))))  
print(list(map(lambda x: x * x, range(5))))
```

```
↳ [0, 1, 4, 9, 16]  
   [0, 1, 4, 9, 16]
```

Accumulate, Reduce, Groupby

- <https://docs.python.org/3/library/itertools.html> accumulate, groupby
- <https://docs.python.org/3/library/functools.html> reduce

```
[69] import functools, itertools

x = [1,3,5,6,2]

print(sum(x))
print(functools.reduce(lambda a, b: a+b, x))
print(list(itertools.accumulate(x)))

print(max(x))
print(functools.reduce(lambda a, b: a if a > b else b, x))

for (even, group) in itertools.groupby(x, lambda x: x % 2 == 0):
    print(even, list(group))
```

```
17
17
[1, 4, 9, 15, 17]
6
6
False [1, 3, 5]
True [6, 2]
```

Reduce, Groupby (non-lambda)

- If lambdas were overwhelming
- Reduce is a 2 argument function, groupby is 1 argument boolean function

```
[78] import functools, itertools

x = [1,3,5,6,2]

def add(a, b):
    return a+b
print(functools.reduce(add, x))

def larger(a, b):
    return a if a > b else b
print(functools.reduce(larger, x))

def is_even(x):
    return x % 2 == 0
for (even, group) in itertools.groupby(x, is_even):
    print(even, list(group))
```

```
17
6
False [1, 3, 5]
True [6, 2]
```

Product, Permutations, Combinations



```
from itertools import *  
print(list(product("ABCD", repeat=2)))  
print(list(permutations("ABCD", 2)))  
print(list(combinations("ABCD", 2)))
```

```
[('A', 'A'), ('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'A'), ('B', 'B'), ('B', 'C'), ('B',  
'D'), ('C', 'A'), ('C', 'B'), ('C', 'C'), ('C', 'D'), ('D', 'A'), ('D', 'B'), ('D', 'C'),  
'D'), ('D', 'D')]
```

```
[('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'A'), ('B', 'C'), ('B', 'D'), ('C', 'A'), ('C',  
'B'), ('C', 'D'), ('D', 'A'), ('D', 'B'), ('D', 'C')]
```

```
[('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'C'), ('B', 'D'), ('C', 'D')]
```


Random

- Pseudo-random number generation -> 'pseudo' because everything is a numerical sequence beginning at some **seed**
- **import random as rand**
- **rand.seed(<seed>)** -> is used to set this starting point if we want consistent behaviour each time program is run
- **randint** is single integer, **randrange** can allow you to select integer from dictate consistent range

```
import random as rand
print(rand.randint(5,10))
print(rand.randint(5,10))
print(rand.randint(5,10))
```

```
6
10
10
```

```
import random as rand
rand.seed(12345)
print(rand.randint(5,10))
print(rand.randint(5,10))
print(rand.randint(5,10))
rand.seed(12345)
print(rand.randint(5,10))
print(rand.randint(5,10))
print(rand.randint(5,10))
```

```
8
10
5
8
10
5
```

```
import random as rand
print(rand.randrange(0,10,2))
print(rand.randrange(0,10,2))
print(rand.randrange(0,10,2))
print(rand.randrange(0,10,2))
print(rand.randrange(0,10,2))
```

```
2
4
8
6
```

Random

- **rand.choice()** lets you select from a collection, and **choices()** a collection from a collection (valid to re-select things)
- **rand.shuffle()** will randomly permutate your collection, rand sample is like **choices()** without replacement (select each item once)
- **rand.random()** -> random real number between 0 and 1
- **rand.uniform()** -> random real number in dictate range
- **rand.normalvariate()** -> one of a number of distribution based random real number range selectors

```
import random as rand
print(rand.choice("ABCDEF"))
print(rand.choice("ABCDEF"))
print(rand.choice("ABCDEF"))
print(rand.choice("ABCDEF"))
print(rand.choice("ABCDEF"))
print(rand.choices("ABCDEF", k=5))
```

```
C
B
C
E
D
['A', 'A', 'C', 'D', 'B']
```

```
x = list(range(0,10,1))
rand.shuffle(x)
print(x)
print(rand.sample(list(range(0,10,1)),k=3))
```

```
[2, 8, 4, 7, 1, 0, 9, 5, 3, 6]
[8, 2, 4]
```

```
print(rand.random())
print(rand.uniform(0,100))
print(rand.normalvariate(0,1))
```

```
0.18997137872182035
34.156042577355215
-1.2375996626329344
```

Onward to ... numpy.

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



UNIVERSITY OF
CALGARY