LECTURE 1

Introduction to Python programming language

ABOUT ME

Dr. Mohammad Rifat Ahmmad Rashid
Assistant Professor
Department of Computer Science & Engineering

Phone: 01314447616

rifat.rashid@ewubd.edu

https://fse.ewubd.edu/computer-science-engineering/faculty-view/rifat.rashid

COURSE CONTENT

- Understand and utilize Python;
- Understand basic data types and data structures in Python.
- Introductory concepts of Python, Getting data into Python (Familiarize and load data files into Python, basic data manipulation);
- Sub-setting, Basic statistical calculations;
- Visualize datasets using low-level and high-level plots in Python.
- Concept of Array and matrix, Inference, random sampling;
- Regression, Programming with Python;
- Writing python functions, Control structures.

MAPPING COURSE LEARNING OUTCOMES (CLOS) WITH THE TEACHING-LEARNING & ASSESSMENT STRATEGY

CLO Description	**Domain/Level**	**Assessment Tool**
Understand basic data structures in Python	Understand	Term and Final Examination
Getting data into Python (data import, export)	Analyze	Term and Final Examination
Visualize datasets using plots	Understand	Term and Final Examination
Writing Python functions, Control structures	Apply	Assignment, Term and Final Examination

ABOUT PYTHON

- Development started in the 1980’s by Guido van Rossum.
  - Only became popular in the last decade or so.
- Python 2.x old version , but Python 3.x is the current version of Python.
- Interpreted, very-high-level programming language.
- Supports a multitude of programming paradigms.
  - OOP, functional, procedural, logic, structured, etc.
- General purpose.
  - Very comprehensive standard library includes numeric modules, crypto services, OS interfaces, networking modules, GUI support, development tools, etc.

PHILOSOPHY

From *The Zen of Python* (https://www.python.org/dev/peps/pep-0020/)

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

NOTABLE FEATURES

- Easy to learn.
- Supports quick development.
- Cross-platform.
- Open Source.
- Extensible.
- Embeddable.
- Large standard library and active community.
- Useful for a wide variety of applications.

THE PROGRAMMING CYCLE FOR PYTHON

GETTING STARTED

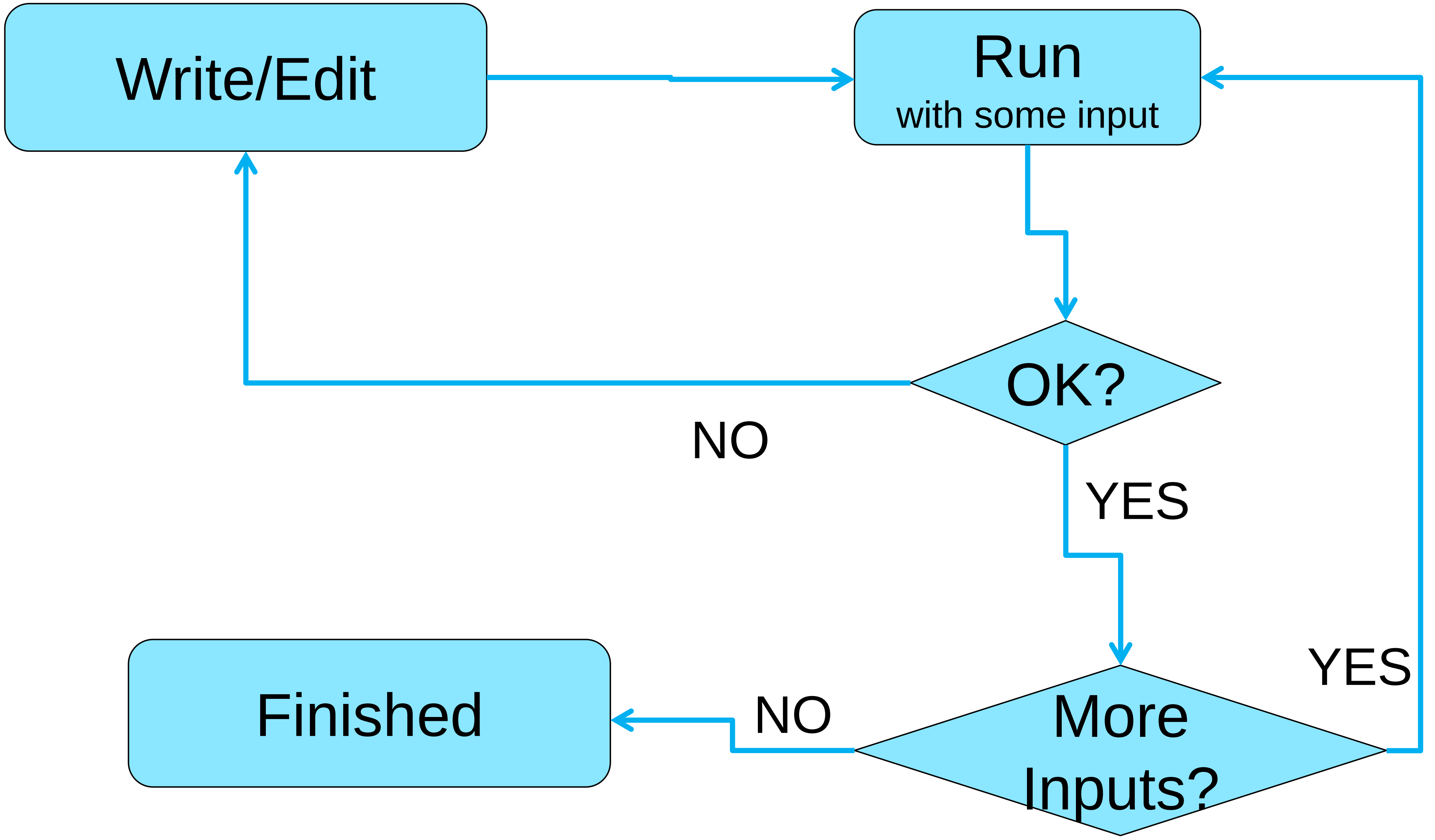
Before we can begin, we need to actually install Python!

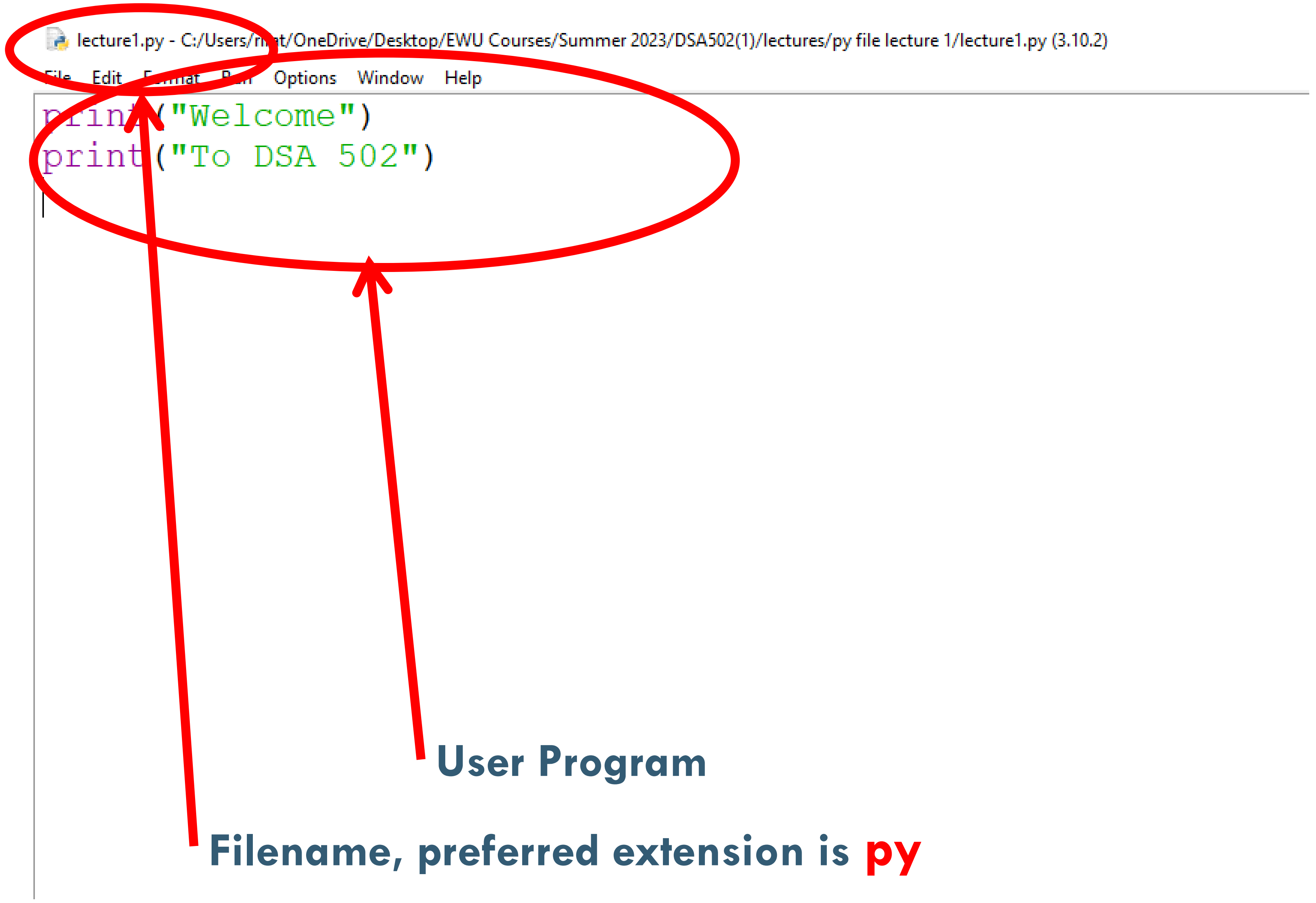
https://www.python.org/download/releases/3.0/
https://docs.anaconda.com/anaconda/install/windows/

Python IDE

https://www.jetbrains.com/pycharm/

Online python editors
https://replit.com/
https://colab.research.google.com/

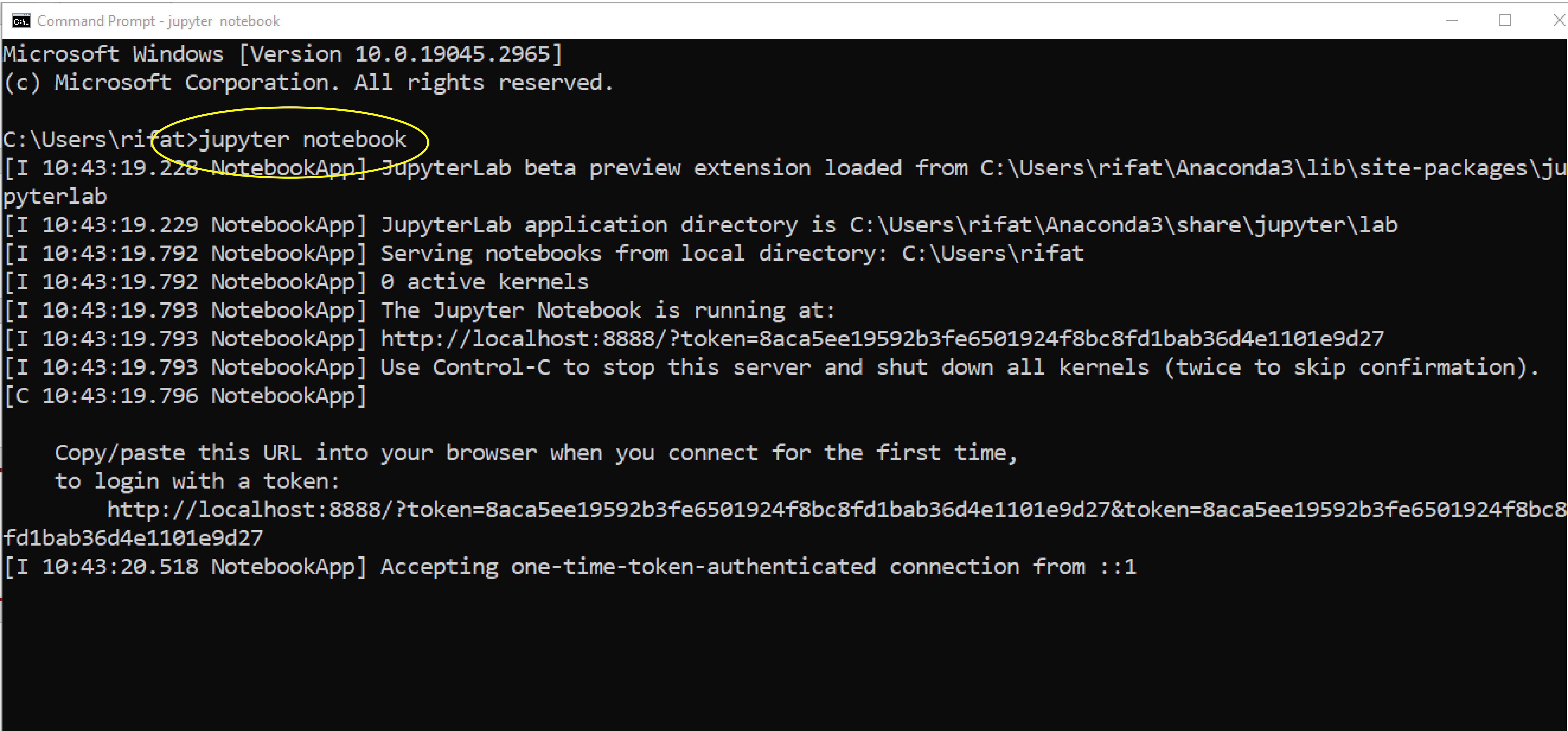

Write/Edit
Run
with some input
OK?
NO
YES
More Inputs?
YES
NO
Finished


lecture1.py - C:/Users/r[illegible]t/OneDrive/Desktop/EWU Courses/Summer 2023/DSA502(1)/lectures/py file lecture 1/lecture1.py (3.10.2)
File Edit Format Run Options Window Help

```
print("Welcome")
print("To DSA 502")
```

User Program

Filename, preferred extension is py



Command Prompt - jupyter notebook

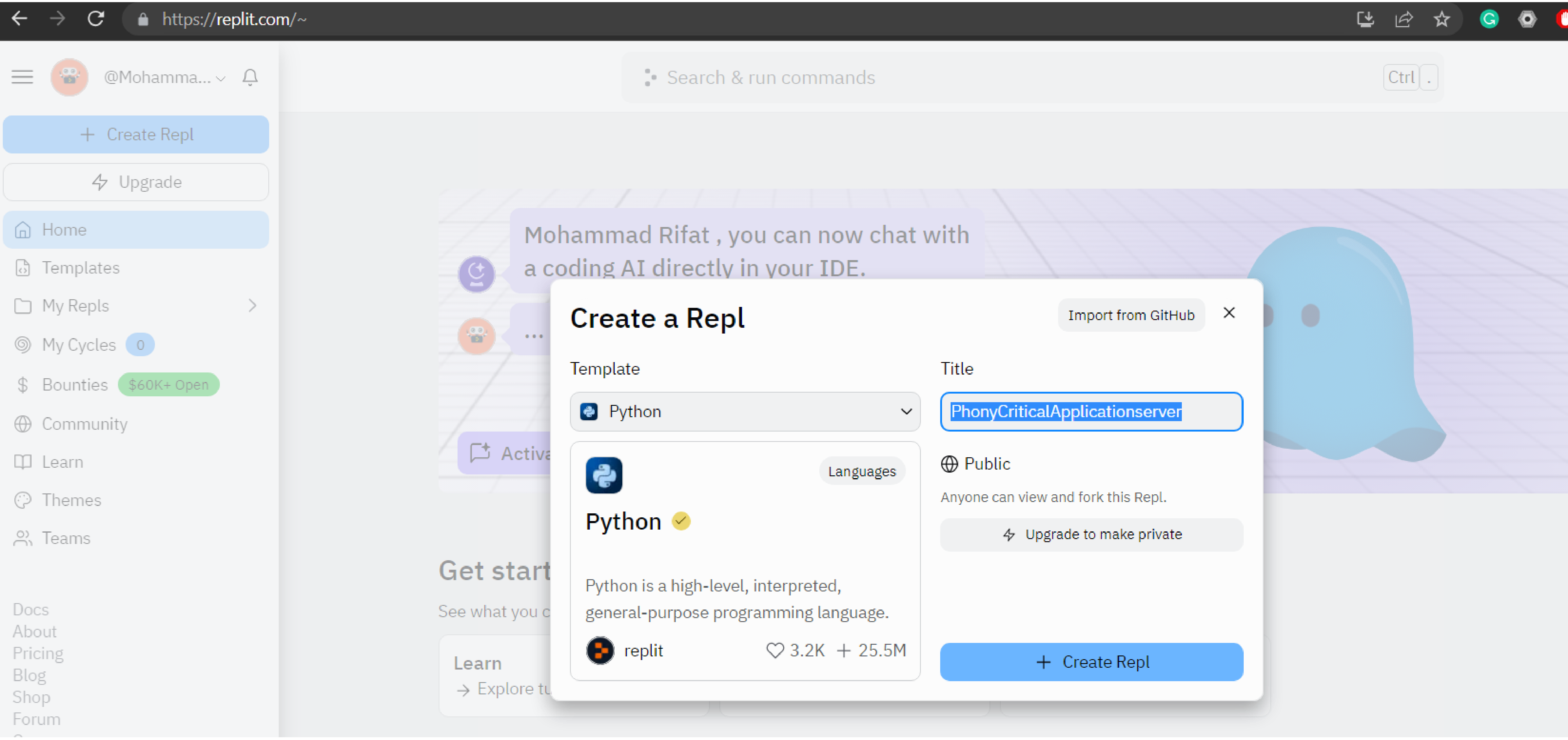
```
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rifat>jupyter notebook
[I 10:43:19.228 NotebookApp] JupyterLab beta preview extension loaded from C:\Users\rifat\Anaconda3\lib\site-packages\ju
pyterlab
[I 10:43:19.229 NotebookApp] JupyterLab application directory is C:\Users\rifat\Anaconda3\share\jupyter\lab
[I 10:43:19.792 NotebookApp] Serving notebooks from local directory: C:\Users\rifat
[I 10:43:19.792 NotebookApp] 0 active kernels
[I 10:43:19.793 NotebookApp] The Jupyter Notebook is running at:
[I 10:43:19.793 NotebookApp] http://localhost:8888/?token=8aca5ee19592b3fe6501924f8bc8fd1bab36d4e1101e9d27
[I 10:43:19.793 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 10:43:19.796 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=8aca5ee19592b3fe6501924f8bc8fd1bab36d4e1101e9d27&token=8aca5ee19592b3fe6501924f8bc8
fd1bab36d4e1101e9d27
[I 10:43:20.518 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

```
IN[1]:
   Welcome
   to Acads
IN[2]:
```

Python Shell Prompt

Outputs

PYTHON SHELL IS INTERACTIVE



Mohammad Rifat , you can now chat with a coding AI directly in your IDE.

Create a Repl

Template: Python

Title: PhonyCriticalApplicationserver

Python

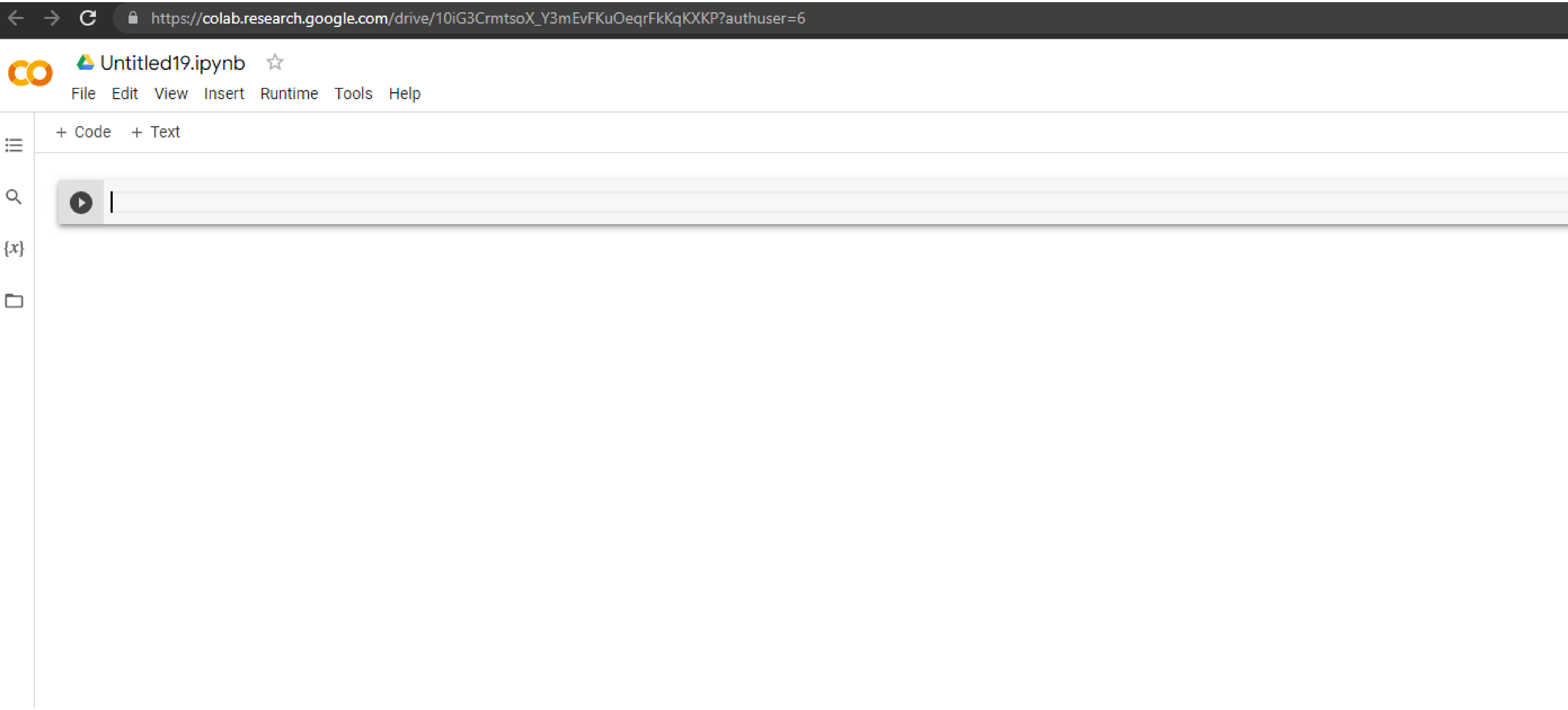
Python is a high-level, interpreted, general-purpose programming language.

replit 3.2K 25.5M

Public

Anyone can view and fork this Repl.

https://colab.research.google.com/


https://colab.research.google.com/drive/10iG3CrmtsoX_Y3mEvFKuOeqrFkKqKXKP?authuser=6
Untitled19.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
+ Code
+ Text

INTERACTING WITH PYTHON PROGRAMS

Python program communicates its results to user using print

Most useful programs require information from users

- Name and age for a travel reservation system

Python 3 uses input to read user input as a string (str)

FIRST PYTHON PROGRAM

```
""" First Python Program Just to see if we can run
python3 successfully """
name = “Rifat"
print(“Welcome "+name+"\n")
```

INPUT

Take as argument a string to print as a prompt

Returns the user typed value as a string

- details of how to process user string later

```
IN[1]: age =      input('How old are you?')


IN[2]:


IN[3]:
```

ELEMENTS OF PYTHON

A Python program is a sequence of **definitions** and **commands (statements)**

Commands manipulate **objects**

Each object is associated with a **Type**

Type:

- A set of values
- A set of operations on these values

Expressions: An operation (combination of objects and **operators**)

TYPES IN PYTHON

int

- Bounded integers, e.g. 732 or -5

float

- Real numbers, e.g. 3.14 or 2.0

long

- Long integers with unlimited precision

str

- Strings, e.g. ‘hello’ or ‘C’

TYPES IN PYTHON

Scalar

- Indivisible objects that do not have internal structure
- **int** (signed integers), **float** (floating point), **bool** (Boolean), ***NoneType***
  - NoneType is a special type with a single value
  - The value is called **None**

Non-Scalar

- Objects having internal structure
- **str** (strings)

Strings

In [ ]:
```python
my_string = "Python is my favorite programming language!"
```

In [ ]:
```python
my_string
```

In [ ]:
```python
type(my_string)
```

In [ ]:
```python
len(my_string)
```

Respecting PEP8 with long strings

In [ ]:
```python
long_story = (
    "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
    "Pellentesque eget tincidunt felis. Ut ac vestibulum est."
    "In sed ipsum sit amet sapien scelerisque bibendum. Sed "
    "sagittis purus eu diam fermentum pellentesque."
)
long_story
```

`str.replace()`

If you don't know how it works, you can always check the `help`:

```
In [ ]: help(str.replace)
```

This will not modify `my_string` because replace is not done in-place.

```
In [ ]: my_string.replace("a", "?")
        print(my_string)
```

You have to store the return value of `replace` instead.

```
In [ ]: my_modified_string = my_string.replace("is", "will be")
        print(my_modified_string)
```

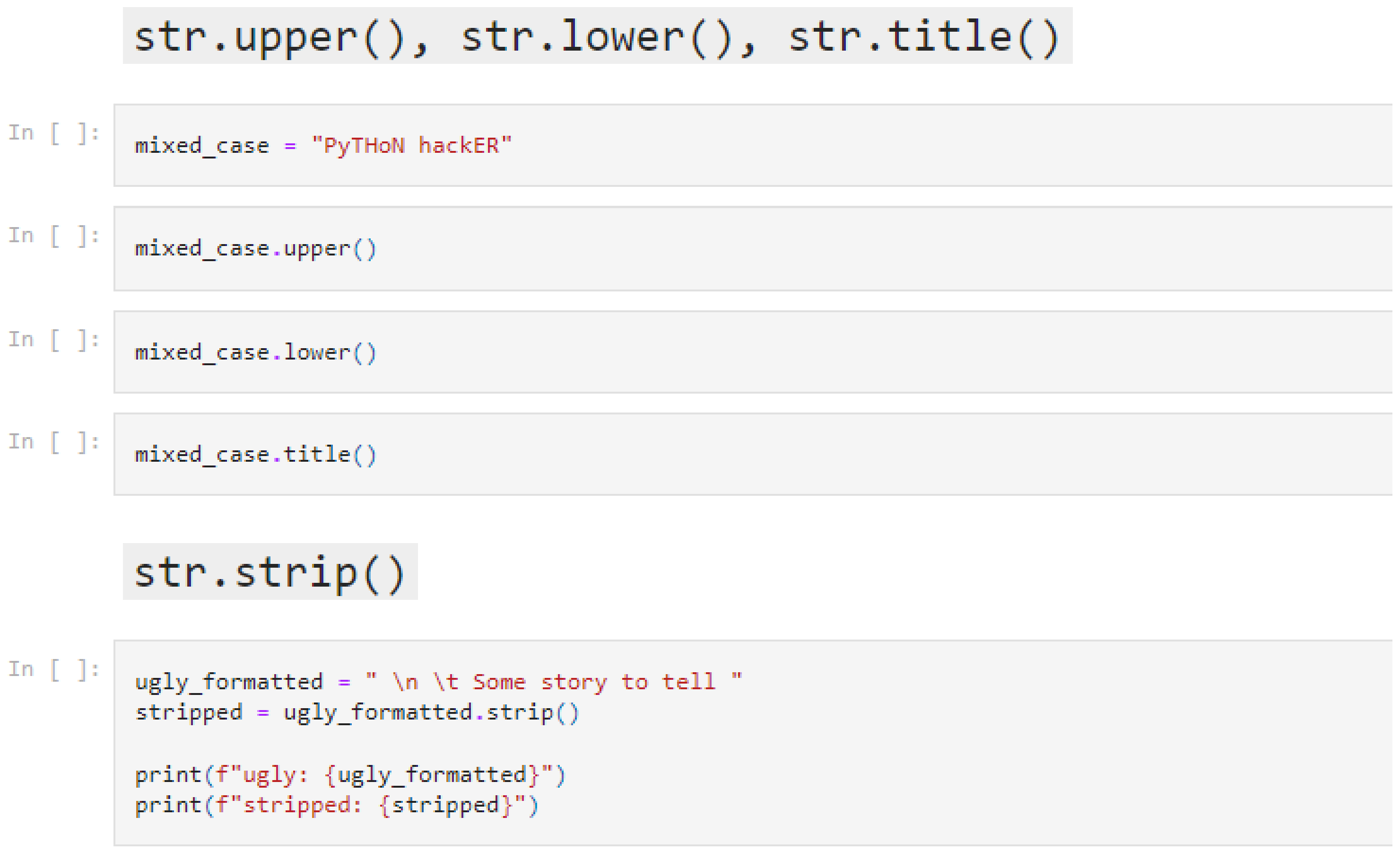
`str.join()`

```
In [ ]:
pandas = "pandas"
numpy = "numpy"
requests = "requests"
cool_python_libs = ", ".join([pandas, numpy, requests])
```

```
In [ ]:
print(f"Some cool python libraries: {cool_python_libs}")
```

Alternative (not as Pythonic and slower):

```
In [ ]:
cool_python_libs = pandas + ", " + numpy + ", " + requests
print(f"Some cool python libraries: {cool_python_libs}")

cool_python_libs = pandas
cool_python_libs += ", " + numpy
cool_python_libs += ", " + requests
print(f"Some cool python libraries: {cool_python_libs}")
```



str.upper(), str.lower(), str.title()

In [ ]:
```
mixed_case = "PyTHoN hackER"
```

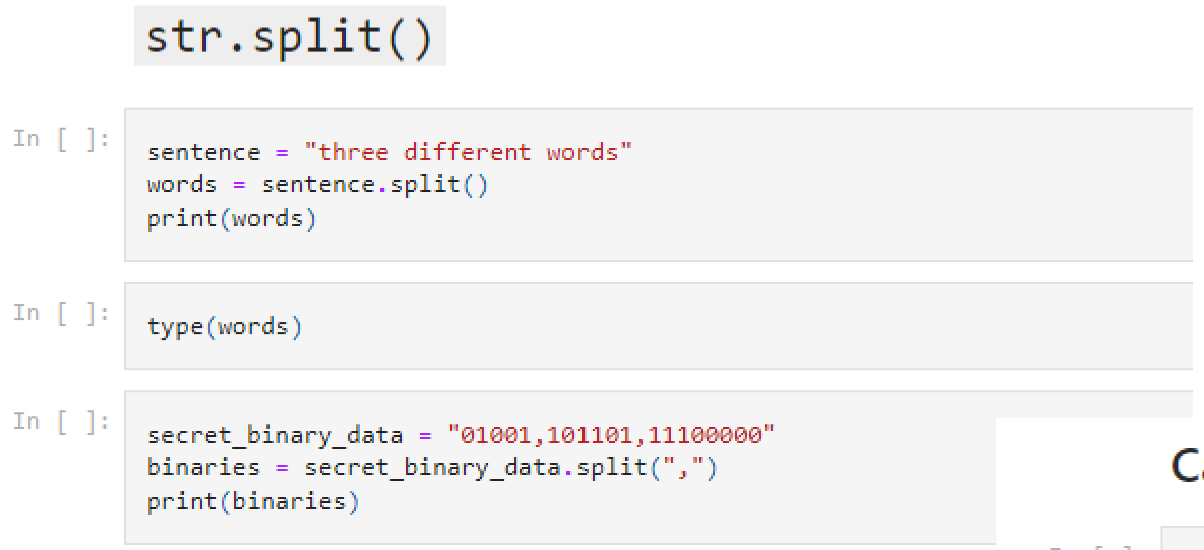
In [ ]:
```
mixed_case.upper()
```

In [ ]:
```
mixed_case.lower()
```

In [ ]:
```
mixed_case.title()
```

str.strip()

In [ ]:
```
ugly_formatted = " \n \t Some story to tell "
stripped = ugly_formatted.strip()

print(f"ugly: {ugly_formatted}")
print(f"stripped: {stripped}")
```

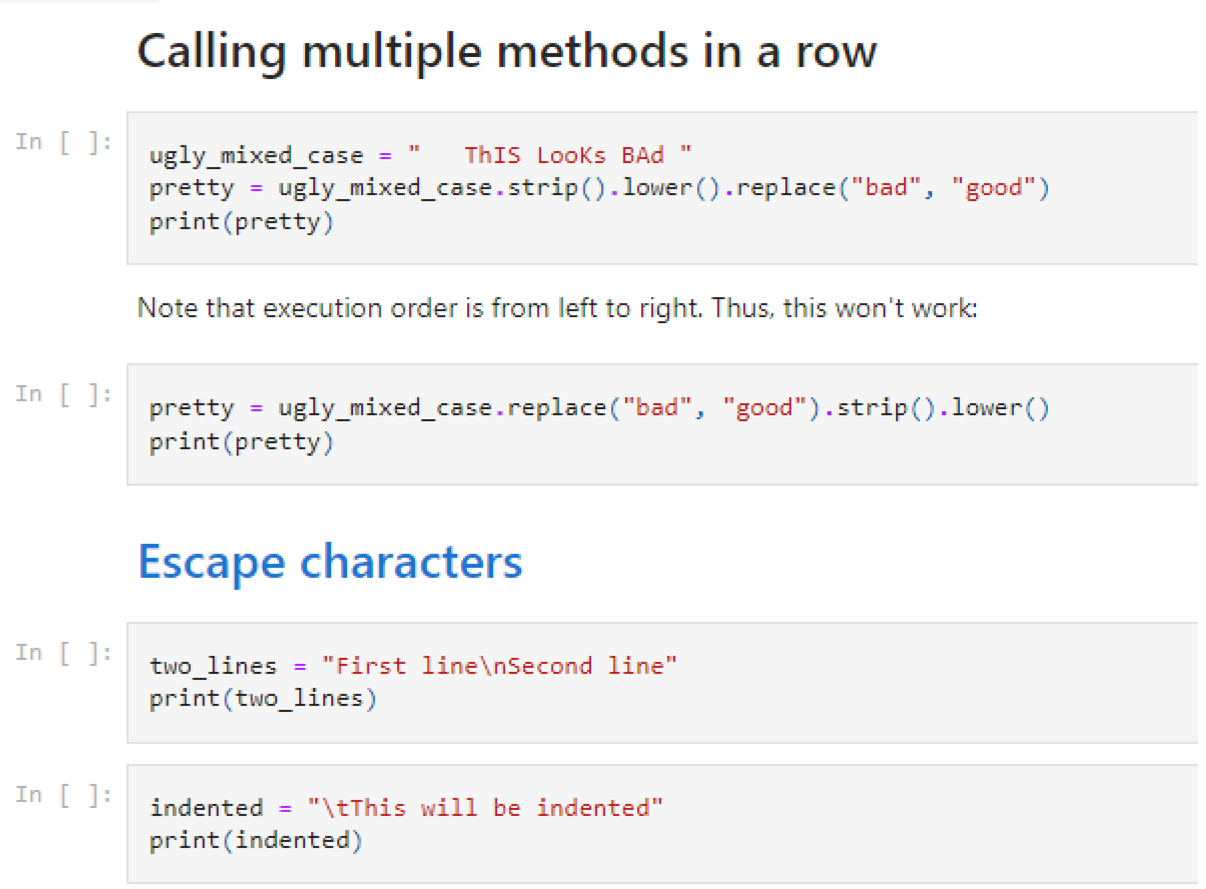


`str.split()`

```
In [ ]:
sentence = "three different words"
words = sentence.split()
print(words)
```

```
In [ ]:
type(words)
```

```
In [ ]:
secret_binary_data = "01001,101101,11100000"
binaries = secret_binary_data.split(",")
print(binaries)
```



Calling multiple methods in a row

```
In [ ]:
ugly_mixed_case = "   ThIS LooKs BAd "
pretty = ugly_mixed_case.strip().lower().replace("bad", "good")
print(pretty)
```

Note that execution order is from left to right. Thus, this won't work:

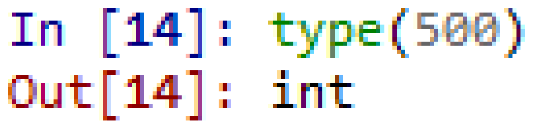
```
In [ ]:
pretty = ugly_mixed_case.replace("bad", "good").strip().lower()
print(pretty)
```

Escape characters

```
In [ ]:
two_lines = "First line\nSecond line"
print(two_lines)
```

```
In [ ]:
indented = "\tThis will be indented"
print(indented)
```

EXAMPLE OF TYPES



```
In [14]: type(500)
Out[14]: int
```

TYPE CONVERSION (TYPE CAST)

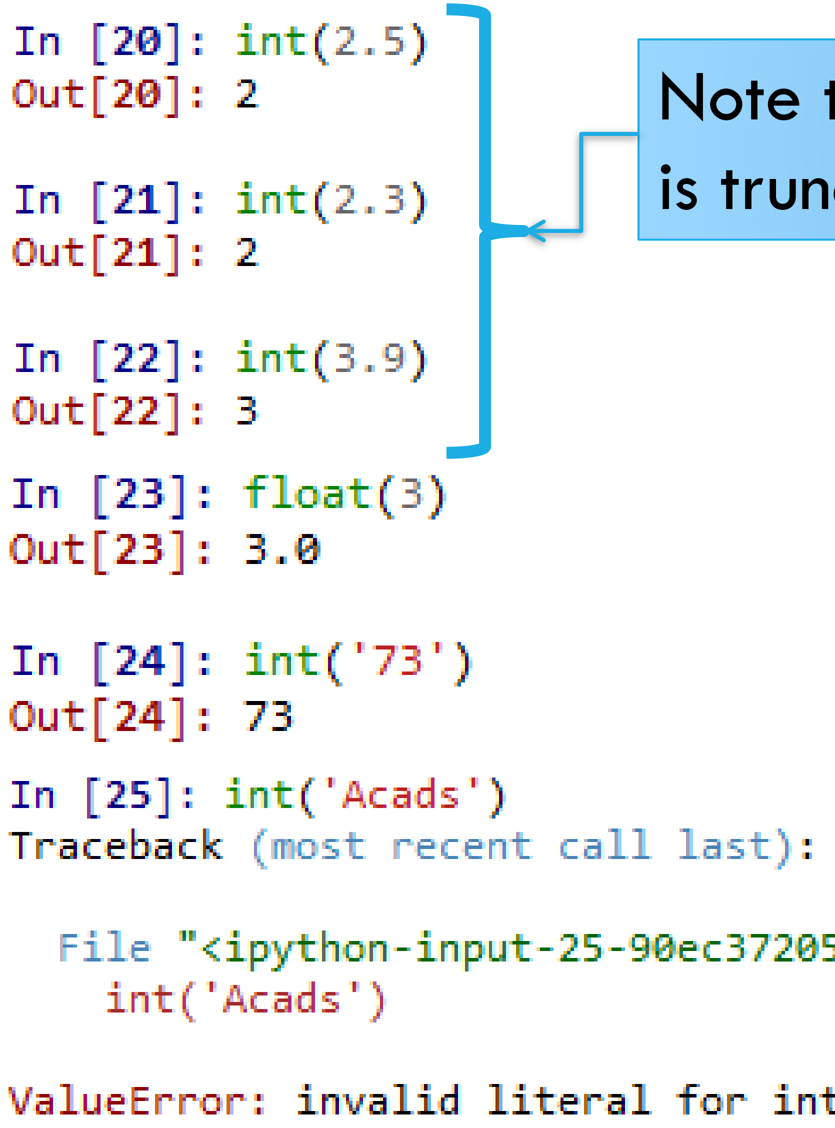
Conversion of value of one type to other

We are used to int ↔ float conversion in Math

- Integer 3 is treated as float 3.0 when a real number is expected
- Float 3.6 is truncated as 3, or rounded off as 4 for integer contexts

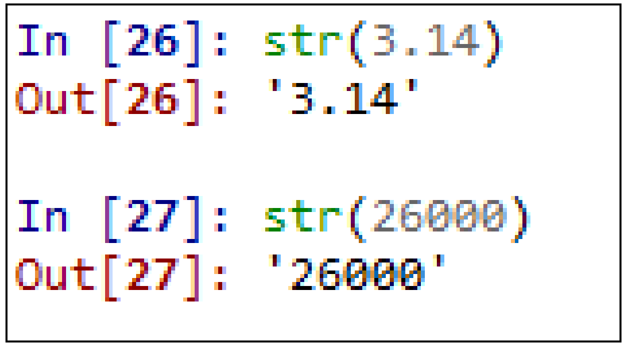
Type names are used as type converter functions

TYPE CONVERSION EXAMPLES



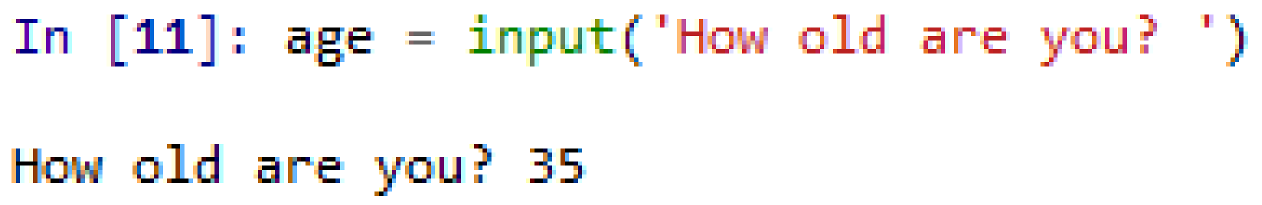
```
In [20]: int(2.5)
Out[20]: 2

In [21]: int(2.3)
Out[21]: 2

In [22]: int(3.9)
Out[22]: 3
In [23]: float(3)
Out[23]: 3.0

In [24]: int('73')
Out[24]: 73
In [25]: int('Acads')
Traceback (most recent call last):

  File "<ipython-input-25-90ec37205222>", line 1, in <module>
    int('Acads')

ValueError: invalid literal for int() with base 10: 'Acads'
```

Note that float to int conversion is truncation, not rounding off

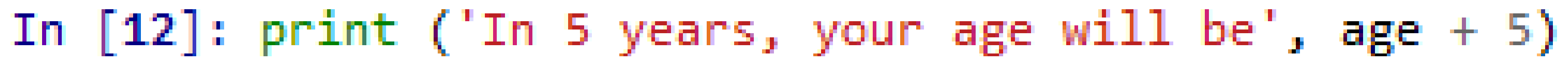


```
In [26]: str(3.14)
Out[26]: '3.14'

In [27]: str(26000)
Out[27]: '26000'
```

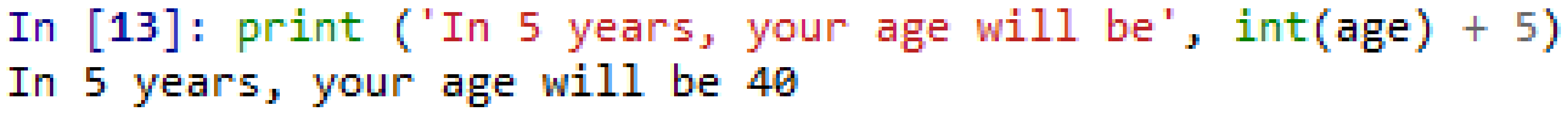
TYPE CONVERSION AND INPUT



```
In [11]: age = input('How old are you? ')

How old are you? 35
```



```
In [12]: print ('In 5 years, your age will be', age + 5)
```



```
In [13]: print ('In 5 years, your age will be', int(age) + 5)
In 5 years, your age will be 40
```

OPERATORS

Arithmetic	+	-	*	//	/	%	**	
Comparison	==	!=	>	<	>=	<=		
Assignment	=	+=	-=	*=	//=	/=	%=	**=
Logical	and	or	not					
Bitwise	&	|	^	~	>>	<<		
Membership	in	not in						
Identity	is	is not						

VARIABLES

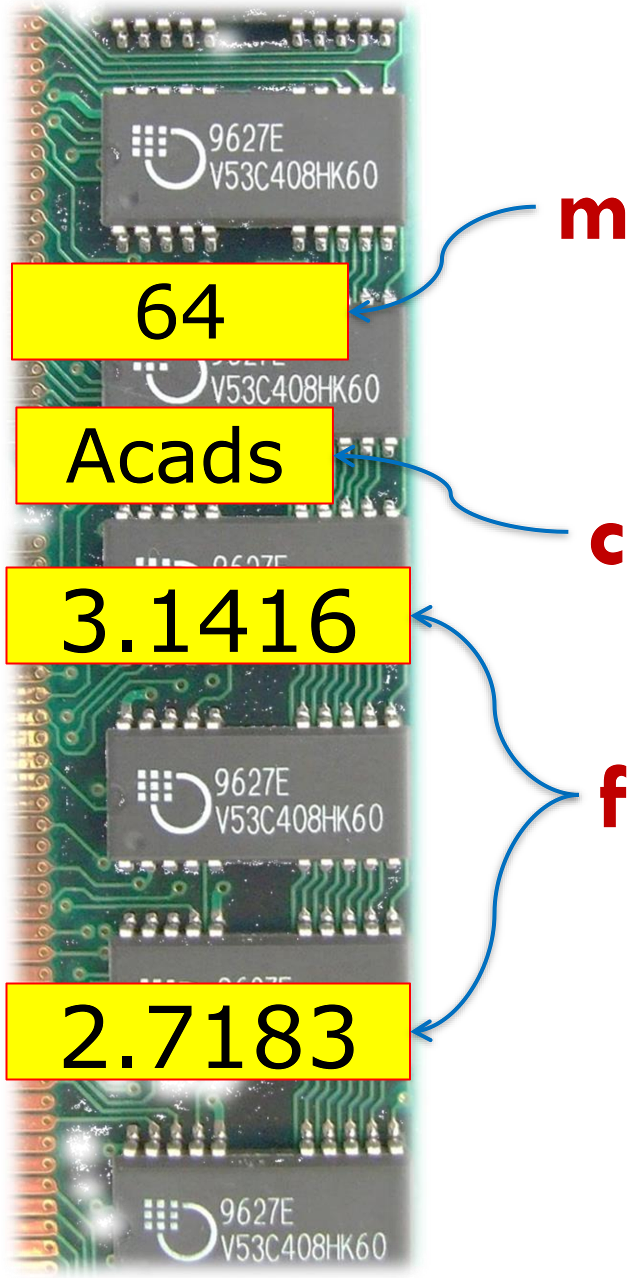
A name associated with an object

Assignment used for binding

```
m = 64;
c = 'Acads';
f = 3.1416;
```

Variables can change their bindings

```
f = 2.7183;
```



ASSIGNMENT STATEMENT

A simple assignment statement

Variable = Expression;

Computes the value (object) of the expression on the right hand side expression (RHS)

Associates the name (variable) on the left hand side (LHS) with the RHS value

= is known as the assignment operator.

MULTIPLE ASSIGNMENTS

Python allows multiple assignments

```
x, y = 10, 20
```

Binds x to 10 and y to 20

Evaluation of multiple assignment statement:

- All the expressions on the RHS of the = are first evaluated **before any binding happens.**
- Values of the expressions are bound to the corresponding variable on the LHS.

```
x, y = 10, 20
x, y = y+1, x+1
```

x is bound to 21 and y to 11 at the end of the program