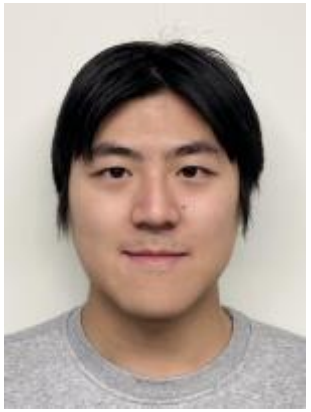
# Introduction to Python for Image Processing

Instructor:
Junyu Chen (Day1 & Day2)
Prof. Marvin Doyley (Day3)

# Self-intro



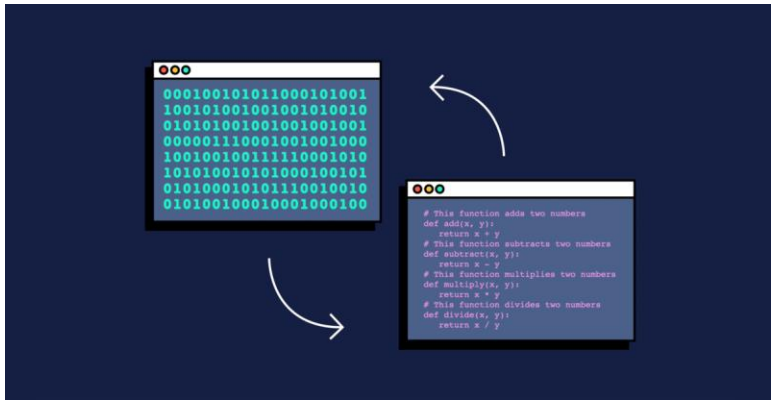
Junyu Chen

- B.S. in Psychology. Peking University
- B.A. in Economics. Peking University
- M.S. in Data Science. University of Rochester
- Incoming Ph.D. student. UR CS


- Email: jchen175@ur.rochester.edu

# What is programming?

- Process of designing, writing, and maintaining code
- Set of instructions for a computer, written using programming languages
- Primary purpose: solve problems and create software
- Automate tasks, process data, develop systems
- Communicate with computers
- Create efficient solutions
- ...




https://www.codecademy.com/resources/blog/programming-languages/

# Why Python?

Python:

```python
print("Hello, world!")
```

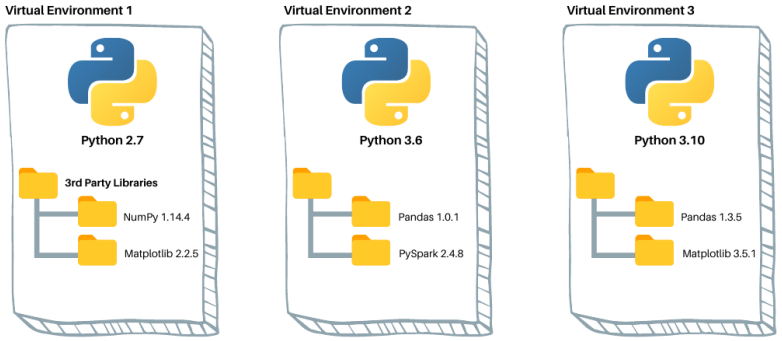
C++:

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

- Easy to learn and readable syntax
- Versatile and multi-purpose
- Extensive libraries and packages
  - collections of pre-written code that can be used to perform specific, so you don't have to start from scratch
    - Web development (Django, Flask)
    - Data analysis (Pandas, NumPy)
    - Machine learning (TensorFlow, PyTorch)
    - Image processing (OpenCV, skimage)
    - ...
- Cross-platform compatibility
- Strong community support
- High demand in the job market

# Python Environment with Anaconda

- Python environment
  - Python interpreter and runtime
  - Collection of packages and libraries
  - Customized settings and configurations

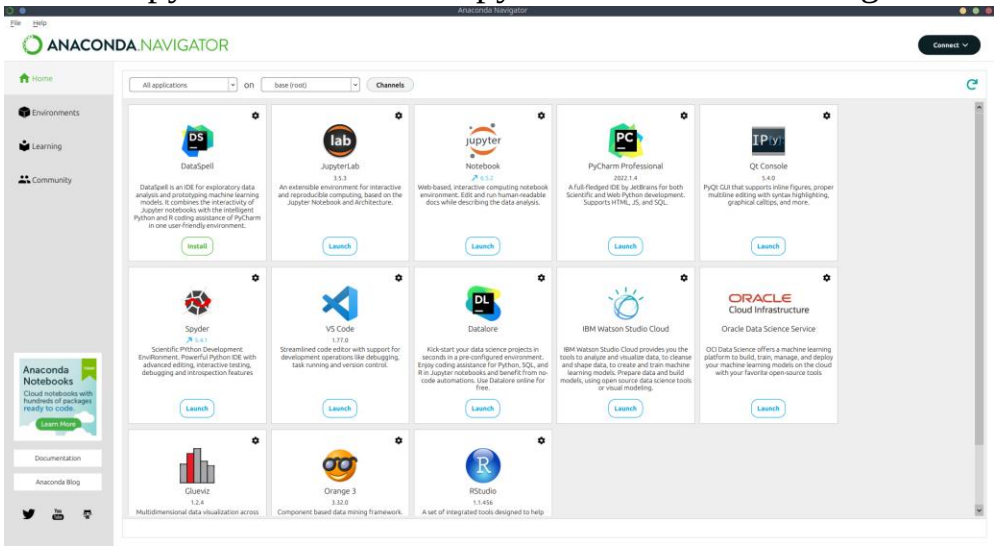



https://www.dataquest.io/blog/a-complete-guide-to-python-virtual-environments/

- Benefits:
  - Dependency management: Each project may have its own specific set of packages and library versions.
  - Isolation: Environments provide isolation, preventing issues that may arise from different projects requiring incompatible versions of the same library or package.
  - Easier reproducibility
  - Easier collaboration

# Python Environment with Anaconda

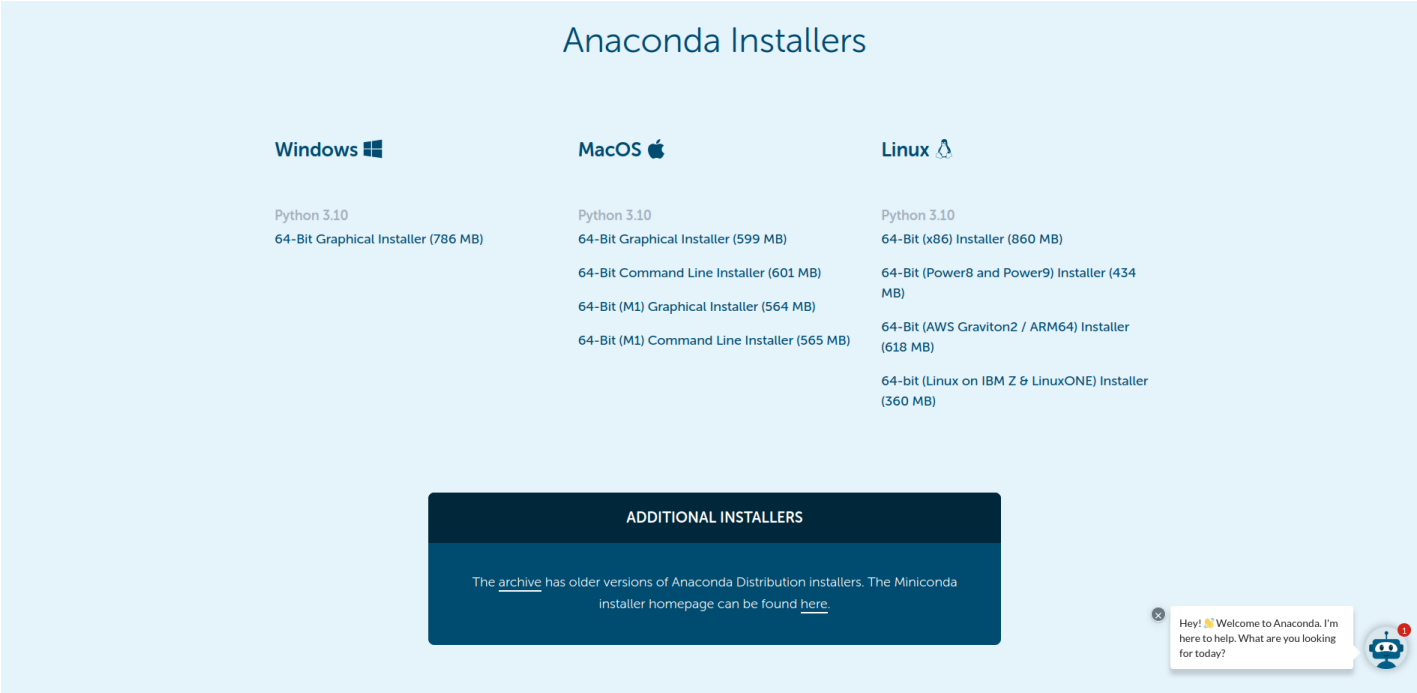
- Anaconda
  - A popular Python distribution that simplifies setting up Python environments, package management, and data science tools.
- Anaconda Navigator
  - A graphical user interface that helps beginners manage environments, packages, and launch applications like Jupyter Notebook and Spyder IDE without needing to use command-line tools.



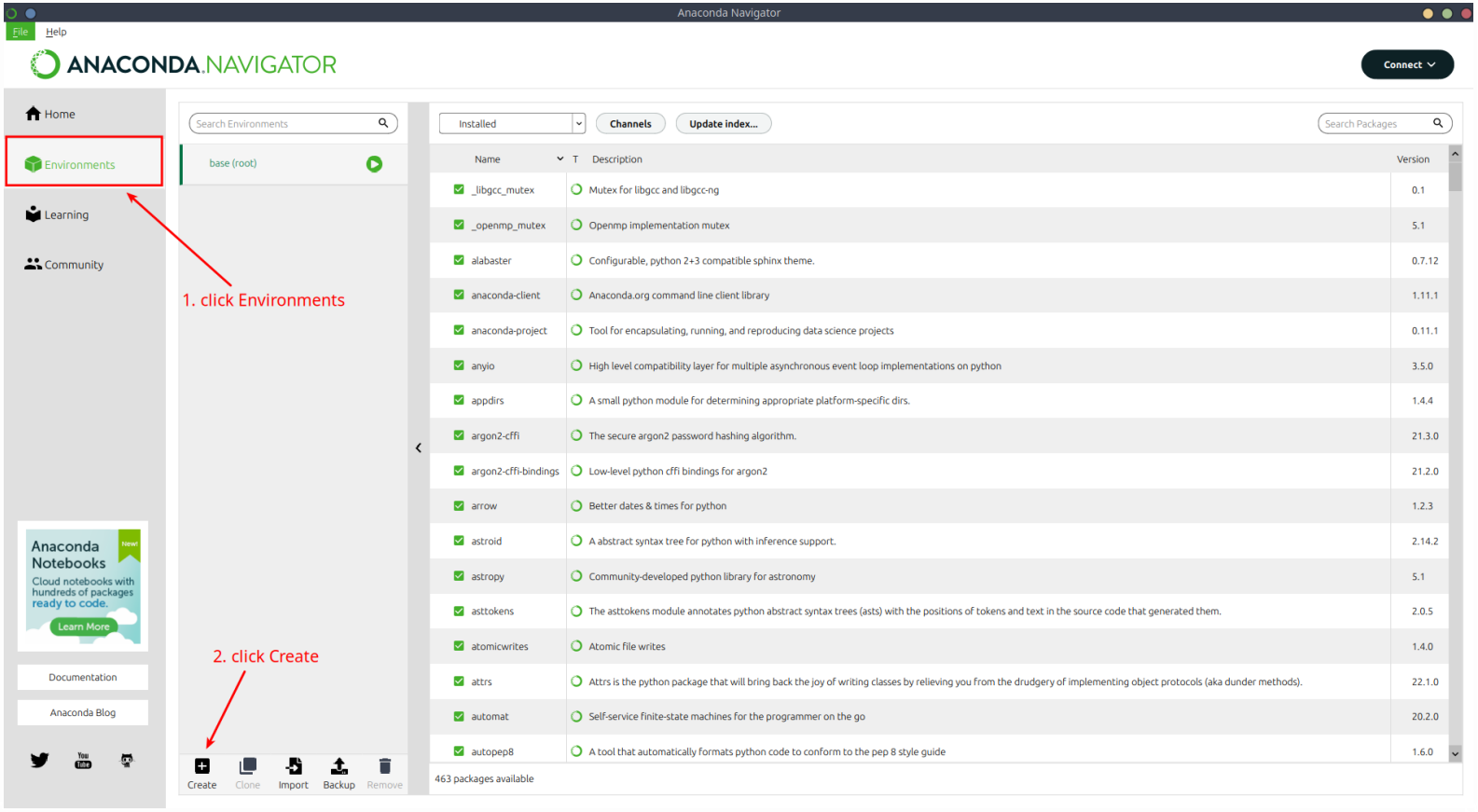

The interface of Anaconda Navigator

# Python Environment with Anaconda

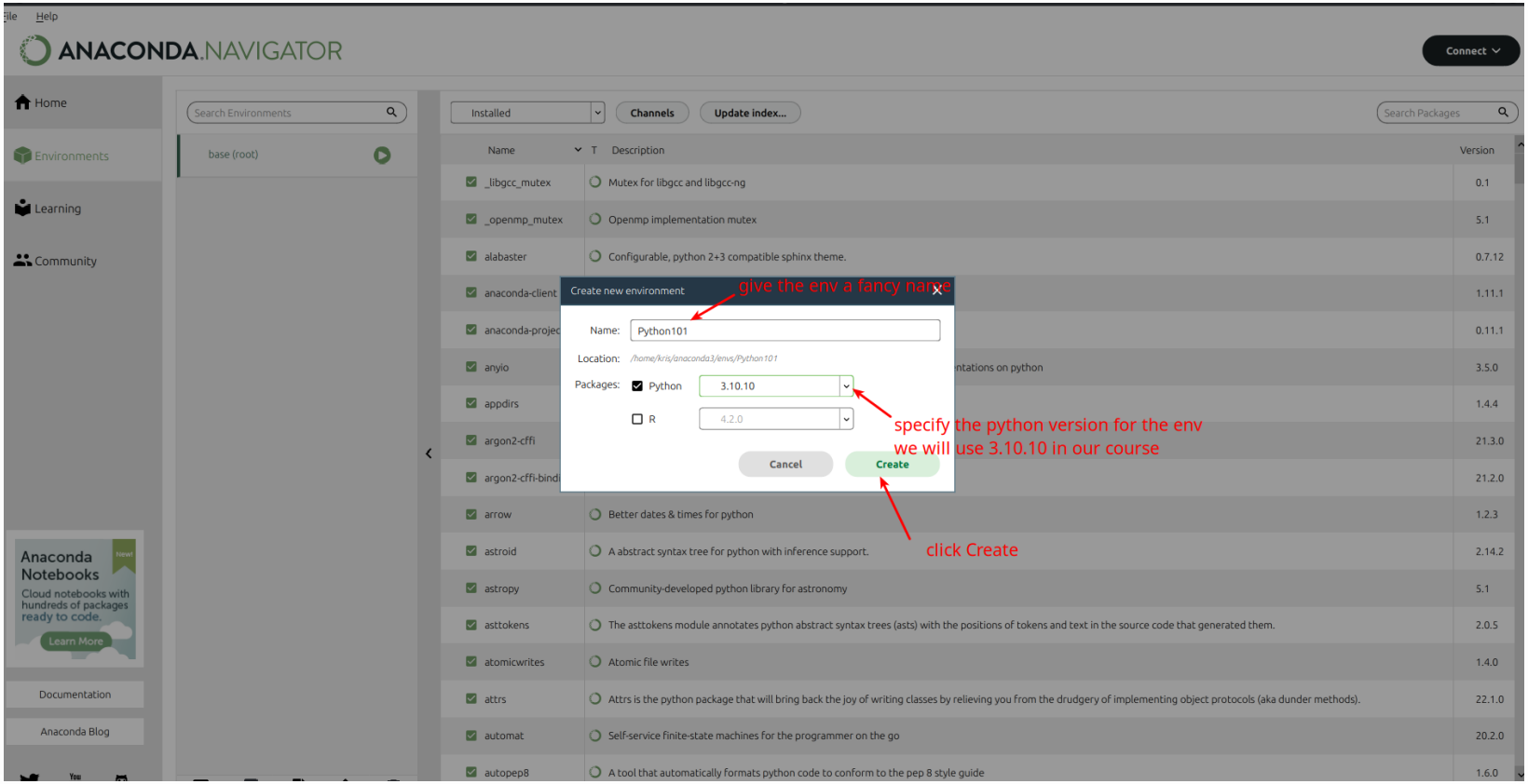
- Install Anaconda Navigator
  - Download link: https://www.anaconda.com/products/distribution#Downloads
  - Tutorial: https://docs.anaconda.com/anaconda/install/



# Python Environment with Anaconda

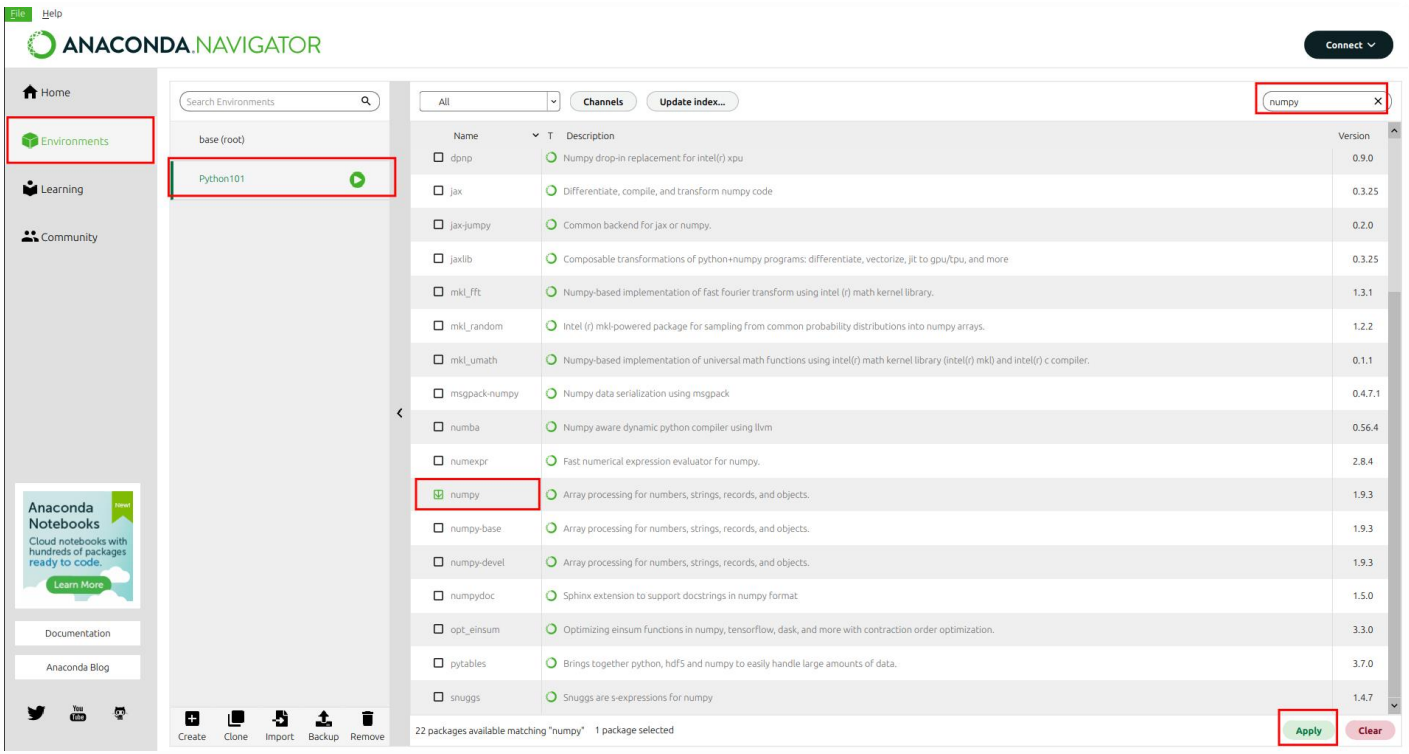


# Python Environment with Anaconda

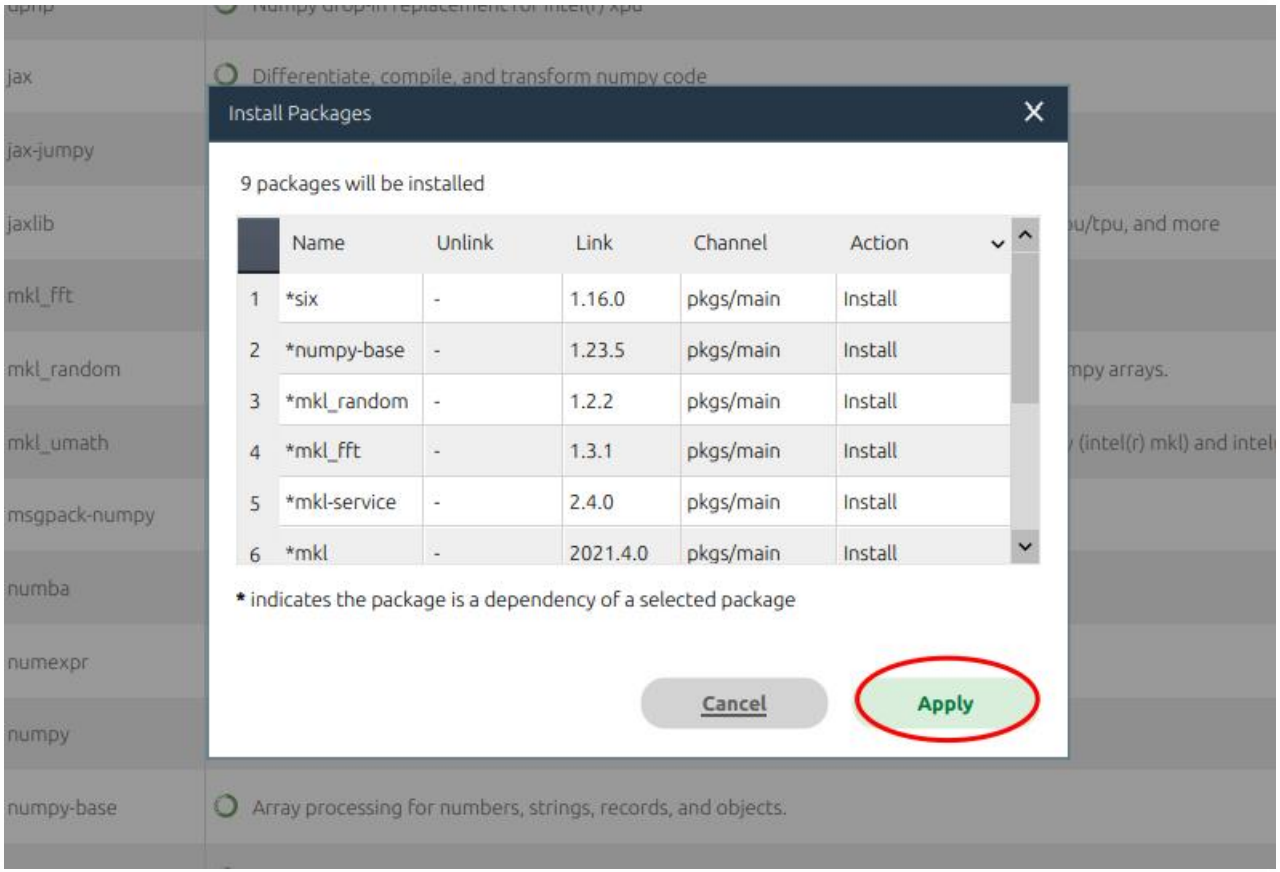


# Python Environment with Anaconda

- Install packages in an environment

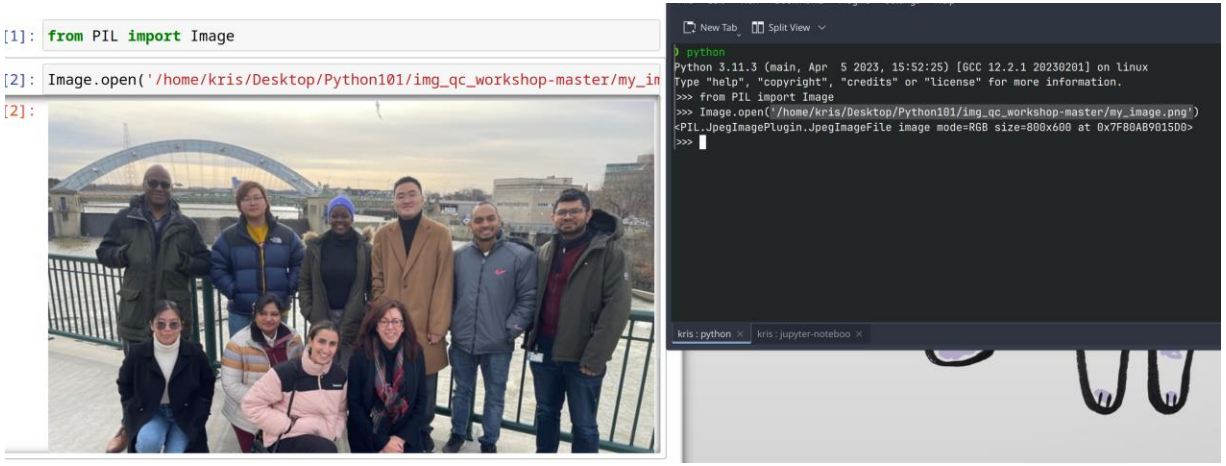


# Python Environment with Anaconda

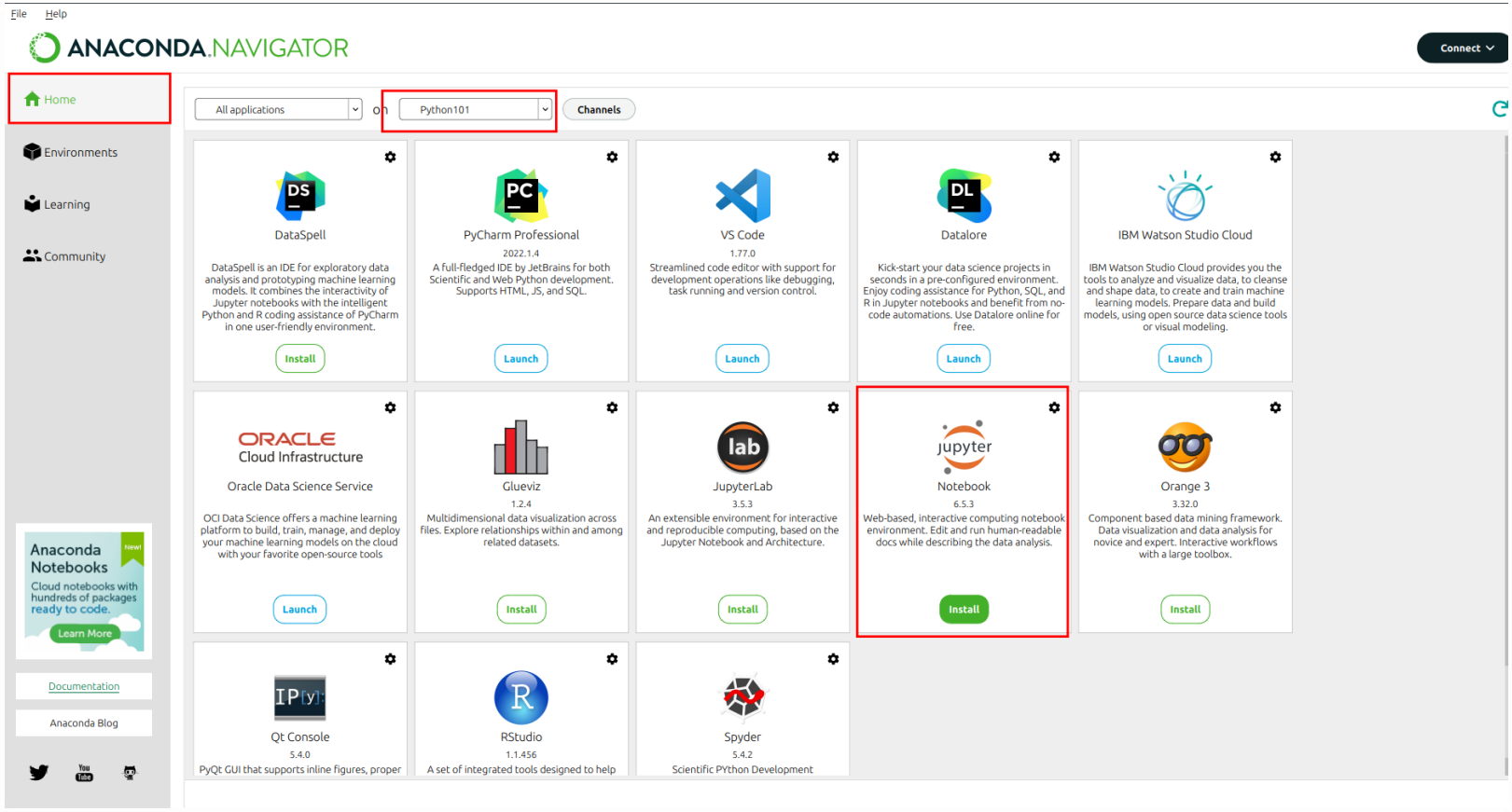


# Python Environment with Anaconda

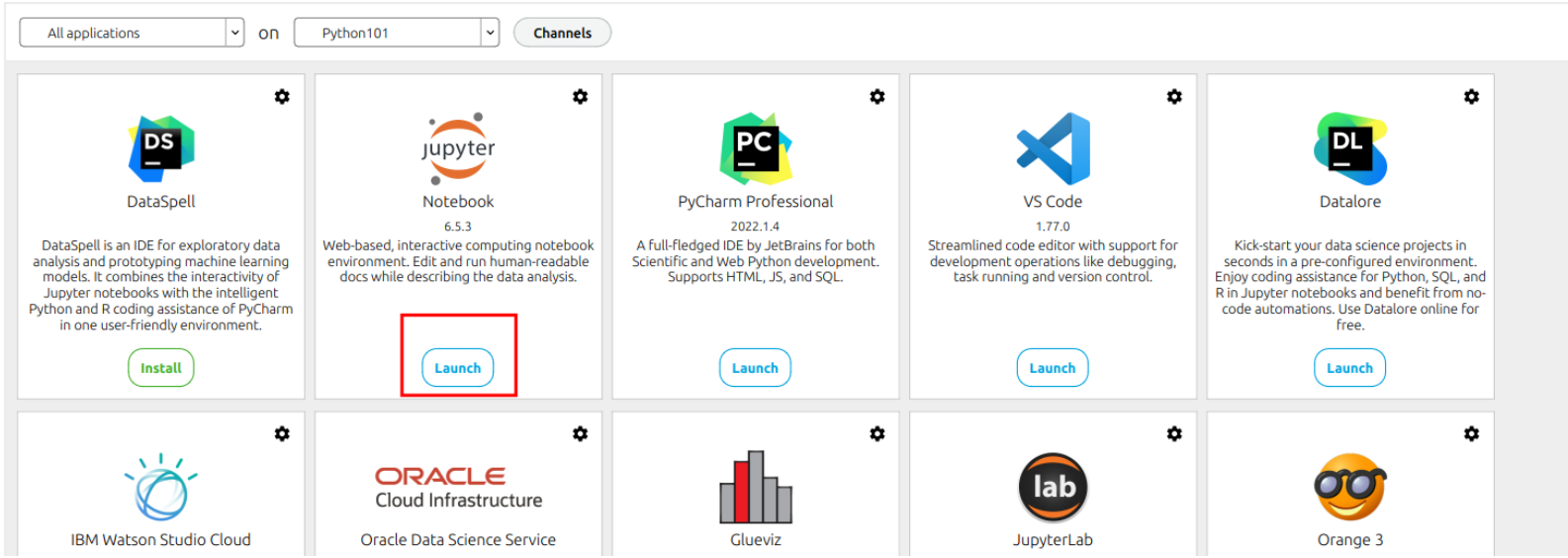
- Jupyter Notebook
  - Interactive coding: write and execute code in an interactive manner
  - Visualization: support the visualization of data using libraries like Matplotlib and Seaborn



# Python Environment with Anaconda



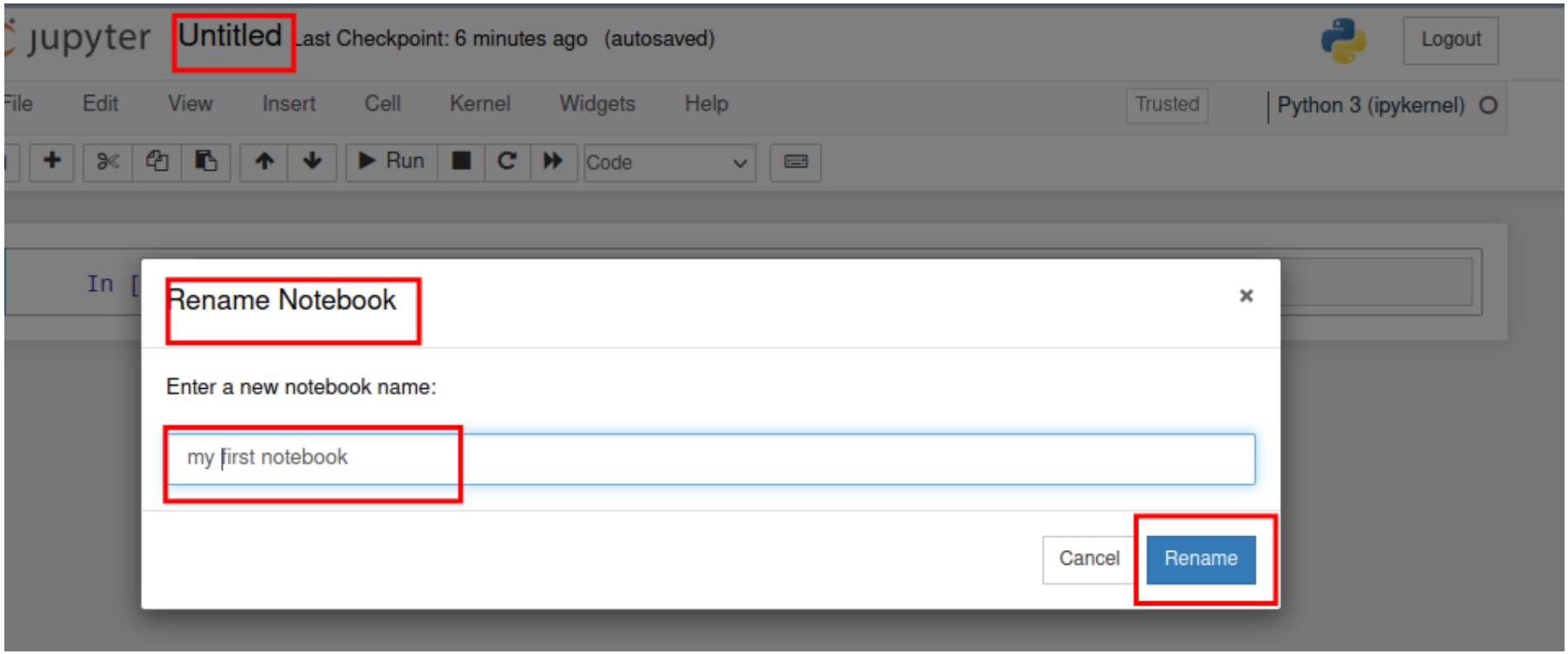
# Python Environment with Anaconda



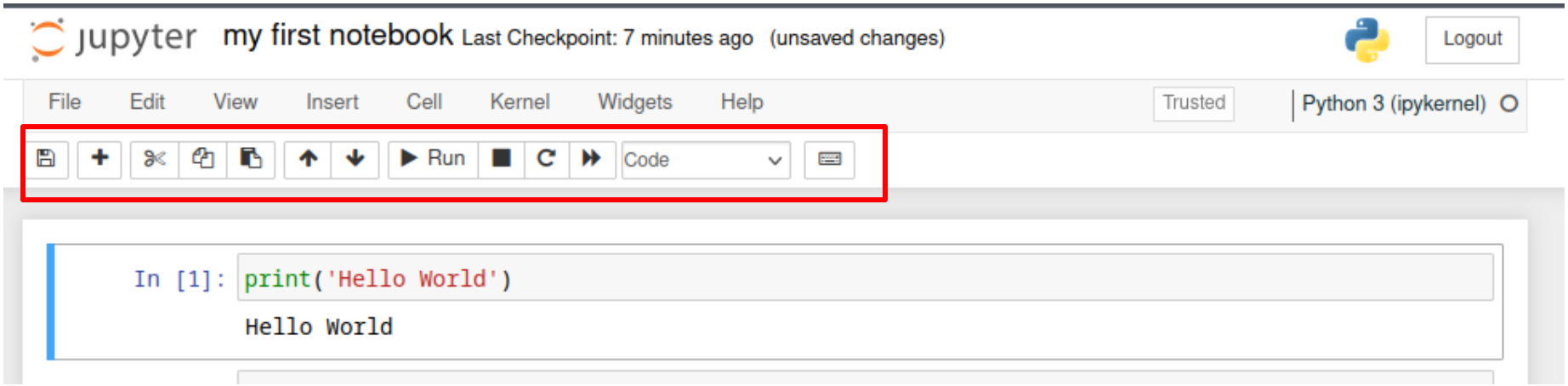
# Python Environment with Anaconda

# Python Environment with Anaconda

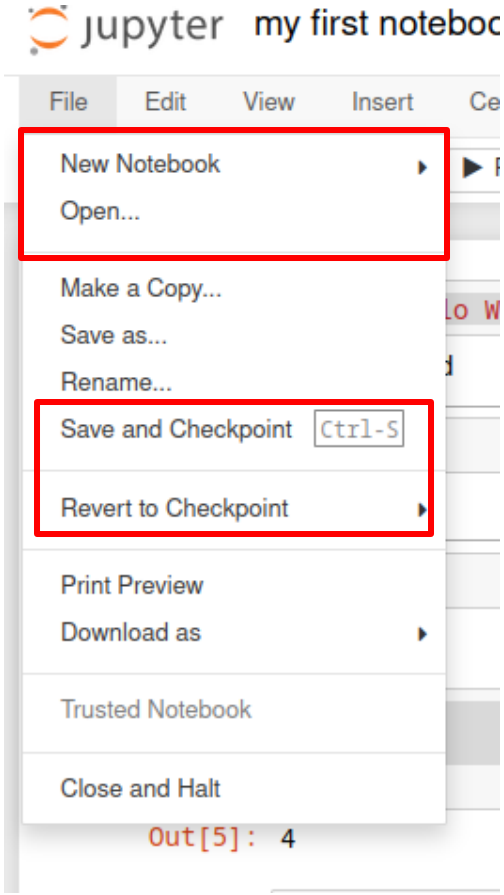
# Python Environment with Anaconda

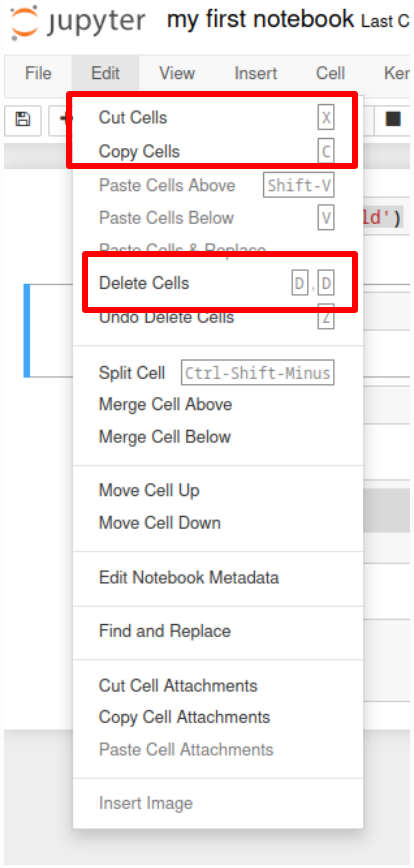


# Python Environment with Anaconda

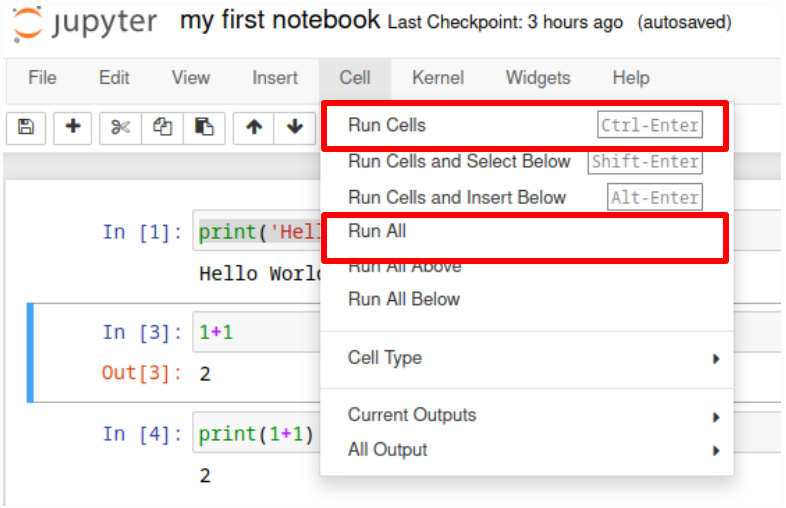


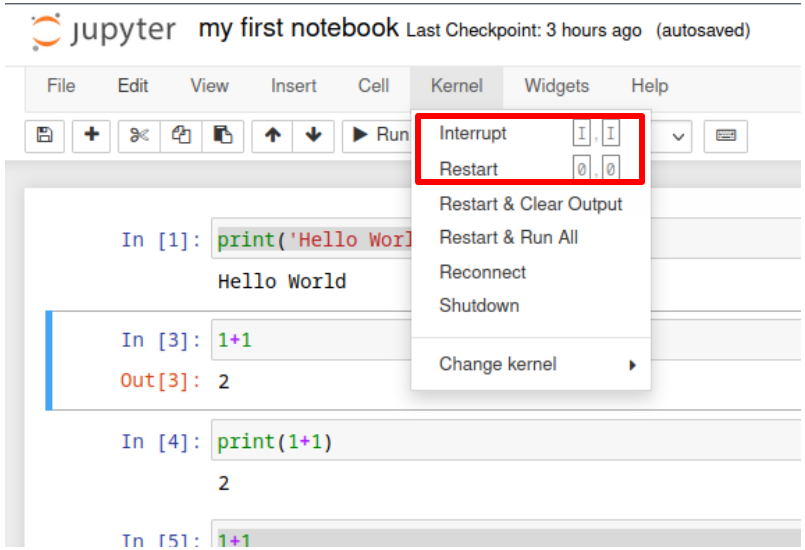
# Python Environment with Anaconda



# Python Environment with Anaconda



# Exercise

- Create a new notebook by clicking on "New" and selecting "Python 3" kernel.
- In the first cell, enter **`print('hello world')`**, then execute the cell by pressing Shift + Enter or by clicking the "Run" button.
- In the second cell, try to get the result of the following equation: 5 + 3 * 2
- Save the checkpoint by clicking on "File" and selecting "Save and Checkpoint" or by using the keyboard shortcut Ctrl + S.
- Delete all cells by selecting "Edit" from the menu and choosing "Delete All Cells".
- To recover to the previous saved checkpoint, click on "File" and select "Revert to Checkpoint". Confirm the revert action when prompted.

# Python Basics

# Primitive Constructs and Syntax

- **primitive constructs**
  - English: words
  - programming language: numbers, strings, simple operators

- **syntax**
  - English:
    - "cat dog boy" □ *not syntactically valid*
    - "cat hugs boy" □ *syntactically valid*
  - Python:
    - "hi"5 □ *not syntactically valid*
    - 3.2*5 □ *syntactically valid*

```
In [11]: "hi"*5
Out[11]: 'hihihihihi'

In [12]: "hi"5
           Input In [12]
             "hi"5
                 ^
SyntaxError: invalid syntax
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Objects

- programs manipulate data objects
- objects have a type that defines the kinds of things programs can do to them
  - Ana is a human so she can walk, speak, etc.
- objects are
  - scalar (cannot be subdivided)
  - non-scalar (have internal structure that can be accessed)

```
In [15]: type([1,2,'apple','banana',5])
Out[15]: list

In [16]: [1,2,'apple','banana',5][2]
Out[16]: 'apple'
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Scalar Objects

- int – represent integers, eg. 5
- float – represent real numbers, eg. 3.27
- bool – represent Boolean values True and False
- NoneType – special and has one value, None

- can use type() to see the type of an object
- can convert object of one type to another
  - float(3) converts integer 3 to float 3.0
  - int(3.9) truncates float 3.9 to integer 3

```
: type(None)
: NoneType

: type(True)
: bool

: type(1)
: int

: type(1.1)
: float
```

```
int(4.7)
```

```
4
```

```
float(3)
```

```
3.0
```

```
type(float(3))
```

```
float
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Printing to Console

- to show output from code to a user, use print command

```
In [11]: 3+2
Out[11]: 5
```

"Out" tells you it's an interaction within the shell only

```
In [1]: 1+1
        1+2
        1+3

Out[1]: 4
```

No "Out" means it is actually shown to a user, apparent when you edit/run files

```
In [2]: print(1+1)
        print(1+2)
        print(1+3)

        2
        3
        4
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Expressions

- **combine objects and operators** to form expressions
- an expression has a **value**, which has a **type**
- **syntax** for a simple expression

  <object> <operator> <object>

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Operations on ints and floats

- **syntax** for a simple expression
  <object> <operator> <object>

- `i+j` → the **sum**
- `i-j` → the **difference**
- `i*j` → the **product**

if both are ints, result is int
if either or both are floats, result is float

- `i/j` → **division**

result is float

```
print(1+1)
print(type(1+1))
```

```
2
<class 'int'>
```

```
print(1/1)
print(type(1/1))
```

```
1.0
<class 'float'>
```

```
print(1+1.0)
print(type(1+1.0))
```

```
2.0
<class 'float'>
```

```
print(1*1)
print(type(1*1))
```

```
1
<class 'int'>
```

- `i%j` → the **remainder** when `i` is divided by `j`
- `i**j` → `i` to the **power** of `j`

```
print(100/33)
print(100%33)
print(100//33)
```

```
3.0303030303030303
1
3
```

```
print(3**2)
print(2**3)
```

```
9
8
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Simple Operations

- parentheses used to tell Python to do these operations first
- operator precedence without parentheses
  - **
  - *
  - /
  - + and - executed left to right, as appear in expression

```
(1+2*3)**4
```

```
2401
```

```
(1+2)*3**4
```

```
243
```

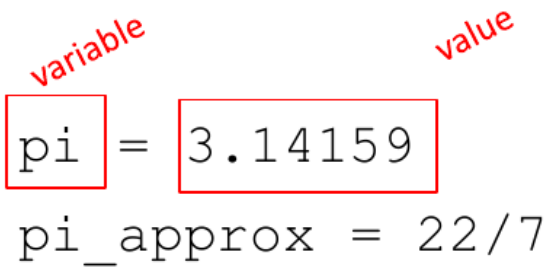
```
1+2*3**4
```

```
163
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Binding Variables and Values

- equal sign is an **assignment** of a value to a variable name




```
pi = 3.14159
print(pi)
```

```
3.14159
```

```
pi_2 = 2*pi
print(pi_2)
```

```
6.28318
```

```
In [32]: 2_pi = 2*pi
           Input In [32]
             2_pi = 2*pi
              ^
SyntaxError: invalid decimal literal
```

- value stored in computer memory
- an assignment binds name to value
- retrieve value associated with name or variable by invoking the name, by typing pi

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Binding Variables and Values

## Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _ )
- Variable names are case-sensitive (age, Age and AGE are three different variables)

```
alpha = 1
Alpha = 2
print(Alpha-alpha)
print(Alpha+alpha)
```

```
1
3
```

## Example

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John"
```

Credit: https://www.w3schools.com/python/gloss_python_variable_names.asp

# Abstracting Expressions

- why give names to values of expressions?
- to **reuse** names instead of values
- easier to change code later

```
pi = 3.14159
radius = 2.2
area = pi*(radius**2)
```

```
pi = 3.14159
radius = 2.2
area = pi*(radius**2)
perimeter = 2*pi*radius

print(area)
print(perimeter)
```

```
15.205295600000001
13.822996
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Changing Bindings

- can re-bind variable names using new assignment statements to **reuse** names instead of values
- previous value may still store in memory but lost the handle for it
- value for area does not change until you tell the computer to do the calculation again

```
pi = 3.14159
radius = 2.2
area = pi*(radius**2)

print(radius)
print(area)
```

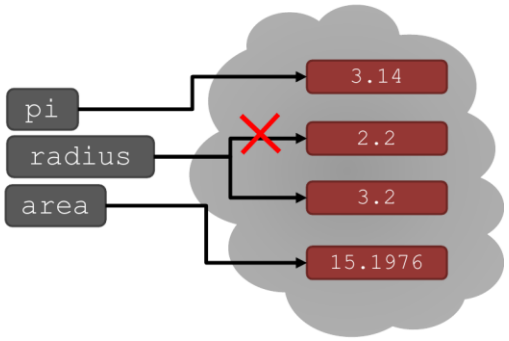
```
2.2
15.205295600000001
```

```
pi = 3.14159
radius = 2.2
area = pi*(radius**2)
radius = radius + 1

print(radius)
print(area)
```

```
3.2
15.205295600000001
```

```
pi = 3.14159
radius = 2.2
area = pi*(radius**2)
radius = radius + 1
area = pi*(radius**2)

print(radius)
print(area)
```

```
3.2
32.169881600000004
```




Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Exercise: Variable Types and Binding

1. Declare a variable called **`my_variable`** and assign an integer value of your choice to it.
2. Print the value of **`my_variable`** to the console.
3. Reassign a different value of a different data type to **`my_variable`**.
4. Print the updated value of **`my_variable`** to the console.
5. Declare another variable called **`new_variable`** and assign a string value to it.
6. Print the value of **`new_variable`** to the console.
7. Swap the values of **`my_variable`** and **`new_variable`** using a *temporary variable.*
8. Print the values of **`my_variable`** and **`new_variable`** after the swap

# String objects

- letters, special characters, spaces, digits
- enclose in quotation marks or single quotes

```
hi_message = 'good morning'
print(hi_message)
```

```
good morning
```

```
hi_message = "good morning"
print(hi_message)
```

```
good morning
```

- concatenate strings
- do some operations on a string as defined in Python docs

```
hi_message = "good morning"
name = "Tom"
print(hi_message + name)
print(hi_message + ', ' + name)
print(hi_message + ', ' + name*3)
```

```
good morningTom
good morning, Tom
good morning, TomTomTom
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# INPUT/OUTPUT: print

- used to output stuff to console
- keyword is print

```
x = 1
print(x)
x_str = str(x)
print("my fav num is", x, ".", "x =", x)
print("my fav num is " + x_str + ". " + "x = " + x_str)
```

```
1
my fav num is 1 . x = 1
my fav num is 1. x = 1
```

- f and {}: embed variable into a string

```
current_variable = 1
print("variable value = current_variable")
print(f"variable value = {current_variable}")
```

```
variable value = {current_variable}
variable value = 1
```

Tab: autocomplete

```
: a_variable_with_a_super_long_name = "hi"
a
```

after you hit tab:

```
a_variable_with_a_super_long_name = "hi"
a_variable_with_a_super_long_name
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# INPUT/OUTPUT: input(" ")

- prints whatever is in the quotes
- user types in something and hits enter
- binds that value to a variable

```
In [ ]: input_text = input("type anyting...")
```

```
In [55]: print(f'user typed: {input_text}')
         user typed: happy
```

```
In [*]: input_text = input("type anyting...")
        type anyting... happy
```

- input gives you a **string** so must cast if working with numbers

```
input_number = input("type any number...")
print('type of input_number without casting:', type(input_number))
input_number = float(input_number)
print('type of input_number after casting:', type(input_number))
type any number... 3.14
```

```
]: input_number = input("type any number...")
   print('type of input_number without casting:', type(input_number))
   input_number = float(input_number)
   print('type of input_number after casting:', type(input_number))
   type any number...3.14
   type of input_number without casting: <class 'str'>
   type of input_number after casting: <class 'float'>
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Comments

- explain what a piece of code does, provide context, or add notes to the code for other programmers to understand
- ignored by the Python interpreter, which means they do not affect the execution of the code
- single-line comments start with a hash symbol (#)
- Multi-line comments are enclosed in triple quotes (''' or """)

```
# This is a single-line comment
print("Hello, World!")  # This is another single-line comment
```

```
Hello, World!
```

```
"""
This is a multi-line comment.
It can span multiple lines.
"""

print("Hello, World!")
```

```
Hello, World!
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Comparison operators on int, float, string

- i and j are variable names
- comparisons below evaluate to a Boolean

```
i > j
i >= j
i < j
i <= j
```

```
In [7]: 'apple'=='banana'
Out[7]: False

In [8]: 1 > 2
Out[8]: False

In [10]: 5.0 >= 2
Out[10]: True
```

`i == j` → **equality** test, `True` if `i` is the same as `j`

`i != j` → **inequality** test, `True` if `i` not the same as `j`

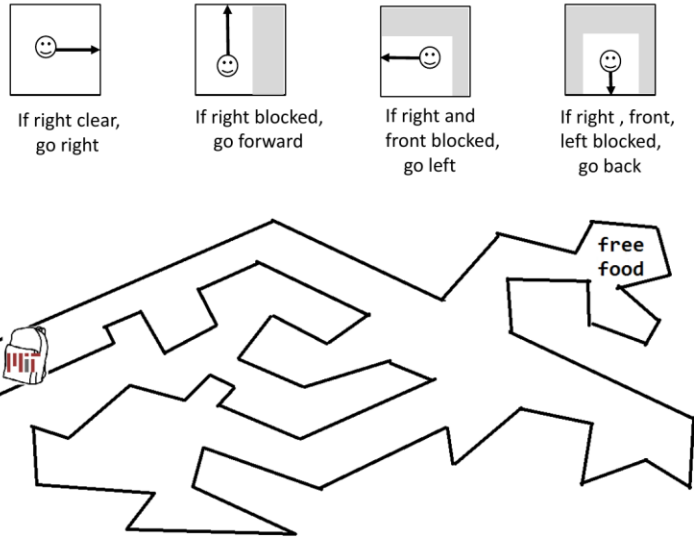
Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Exercise: Comparison and String Operations

1. Prompt the user to enter two numbers (**num1** and **num2**) of any data type (integers or floats).
2. Using the input values, perform the following tasks:
    - Print the sum of **num1** and **num2**.
    - Check if **num1** is equal to **num2** and print the result.
    - Check if **num1** is greater than **num2** and print the result.
    - Convert **num1** and **num2** to strings and concatenate them.
    - Print the length of the concatenated string.
3. Prompt the user to enter two strings (**str1** and **str2**).
4. Using the input values, perform the following tasks:
    - Check if **str1** is equal to **str2** and print the result.
    - Check if **str1** is greater than **str2** and print the result.
    - Concatenate **str1** and **str2**.
    - Print the length of the concatenated string.

# Control flow - branching

- <condition> has a value True or False
- evaluate expressions in that block if <condition>is True




```
if <condition>:
    <expression>
    <expression>
    ...
```

```
if <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

```
if <condition>:
    <expression>
    <expression>
    ...
elif <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Control flow - branching

- <condition> has a value True or False
- evaluate expressions in that block if <condition>is True

```
if <condition>:
    <expression>
    <expression>
    ...
```

```
if <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

```
if <condition>:
    <expression>
    <expression>
    ...
elif <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

python Copy code

```python
age = 25
if age >= 18:
    print("You are an adult.")
```

python Copy code

```python
age = 25
if age < 18:
    print("You are a minor.")
elif age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

In [2]:

```
a = input('assign a value for A...')
b = input('assign a value for B...')
a = float(a)
b = float(b)
if a > b:
    print('A is greater than B')
elif a == b:
    print('A is equal to B')
else:
    print('A is smaller than B')
```

```
assign a value for A...312
assign a value for B...2
A is greater than B
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Indentation

- **matters** in Python
- how you denote blocks of code

```
x = float(input("Enter a number for x: "))
y = float(input("Enter a number for y: "))
if x == y:
    print("x and y are equal")
    if y != 0:
        print("therefore, x / y is", x/y)
elif x < y:
    print("x is smaller")
else:
    print("y is smaller")
print("thanks!")
```

python Copy code

```
x = 10

if x > 5:
    print("x is greater than 5")
    print("This statement is inside the if block")

print("This statement is outside the if block")
```

```
In [2]: a = input('assign a value for A...')
        b = input('assign a value for B...')
        a = float(a)
        b = float(b)
        if a > b:
            print('A is greater than B')
        elif a == b:
            print('A is equal to B')
        else:
            print('A is smaller than B')
```

```
assign a value for A...312
assign a value for B...2
A is greater than B
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# = vs ==

```
In [2]: a = input('assign a value for A...')
        b = input('assign a value for B...')
        a = float(a)
        b = float(b)
        if a > b:
            print('A is greater than B')
        elif a == b:
            print('A is equal to B')
        else:
            print('A is smaller than B')

        assign a value for A...312
        assign a value for B...2
        A is greater than B
```

What if x = y here?

get a SyntaxError

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Exercise: Grade Classification

- Write a Python program that prompts the user to **enter** their exam score and determines their grade based on the following conditions:
  - If the score is greater than or equal to 90, print "Grade: A"
  - If the score is between 80 and 89 (inclusive), print "Grade: B"
  - If the score is between 70 and 79 (inclusive), print "Grade: C"
  - If the score is between 60 and 69 (inclusive), print "Grade: D"
  - If the score is less than 60, print "Grade: F"

# Control flow - Loops



- Legend of Zelda – Lost Woods
- keep going right, takes you back to this same screen, stuck in a loop

```
if <exit right>:
    <set background to woods_background>
    if <exit right>:
        <set background to woods_background>
        if <exit right>:
            <set background to woods_background>
            and so on and on and on...
        else:
            <set background to exit_background>
    else:
        <set background to exit_background>
else:
    <set background to exit_background>
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Control flow - while LOOPS

```
while <condition>:
    <expression>
    <expression>
    ...
```

- <condition> evaluates to a Boolean
- if <condition> is True, do all the steps inside the while code block
- check <condition> again
- repeat until <condition> is False

```
: # get the sum of ints between 1 and 5:
  # initialize the first num/sum
  cur_num = 1
  all_sum = 0
  # set condition
  while cur_num <= 5:
      # add numbers up
      all_sum = all_sum + cur_num
      cur_num = cur_num + 1
  print(f'result is: {all_sum}')
```

```
result is: 15
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Exercise: Comparison and String Operations

- Write a Python program that prompts the user to create a new password.
  - The program should then allow the user to input the password up to 5 times.
  - If the entered password matches the newly created password, display a success message (hint: the loop can be terminated with the **`break`** statement).
  - Otherwise, provide feedback indicating the number of attempts remaining.

# Control flow - while and for LOOPS

- iterate through numbers in a sequence

```
# more complicated with while loop
n = 0
while n < 5:
    print(n)
    n = n+1
# shortcut with for loop
for n in range(5):
    print(n)
```

for <variable> in range(<some_num>):
    <expression>
    <expression>
    ...

python Copy code

```
for item in sequence:
    # Code block to be executed for each item
```

python Copy code

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Control flow - for LOOPS

- iterate through numbers in a sequence

```
for <variable> in range(<some_num>):
        <expression>
        <expression>
        ...
```

- each time through the loop, <variable> takes a value
- first time, <variable> starts at the smallest value
- next time, <variable> gets the prev value + 1
- etc.

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# range(start, stop, step)

- default values are start = 0 and step = 1 and optional
- loop until value is **stop - 1**

In [8]:
```
mysum = 0
for i in range(7, 10):
    #7+8+9
    mysum += i
    print(mysum)
```

```
7
15
24
```

In [9]:
```
mysum = 0
for i in range(5, 11, 2):
    #5+7+9
    mysum += i
    print(mysum)
```

```
5
12
21
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# break statement

- immediately exits whatever loop it is in
- skips remaining expressions in code block
- **exits only innermost loop!**

```
while <condition_1>:
    while <condition_2>:
        <expression_a>
        break
        <expression_b>
    <expression_c>
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# break statement

- immediately exits whatever loop it is in
- skips remaining expressions in code block
- exits only innermost loop!

```
mysum = 0
for i in range(5, 11, 2):
    mysum += i
    if mysum == 5:
        break
        #jump out side of the innermost loop
        mysum += 1
print(mysum)
```

```
5
```

```
# print pairs of numbers
for i in range(5):
    for j in range(5):
        # ignore the print statement if i is smaller than j
        if i < j:
            break
        print(i, j)
```

```
0 0
1 0
1 1
2 0
2 1
2 2
3 0
3 1
3 2
3 3
4 0
4 1
4 2
4 3
4 4
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Control flow - while and for LOOPS

## `for` VS `while` LOOPS

`for` loops

- **know** number of iterations
- can **end early** via `break`
- uses a **counter**
- **can rewrite** a `for` loop using a `while` loop

`while` loops

- **unbounded** number of iterations
- can **end early** via `break`
- can use a **counter but must initialize** before loop and increment it inside loop
- **may not be able to rewrite** a `while` loop using a `for` loop

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Exercise: Vowel Counter

- Write a Python program that prompts the user to enter a word or phrase. The program should then count the number of vowels (a, e, i, o, u) in the input and display the result.
  - Using a **`for`** loop
  - Using a **`while`** loop
- Hint:
  - **`for char in string: # Looping through a string:`**
  - **`first_char = string[0]  # Access the first character`**
  - **`char.lower()`**