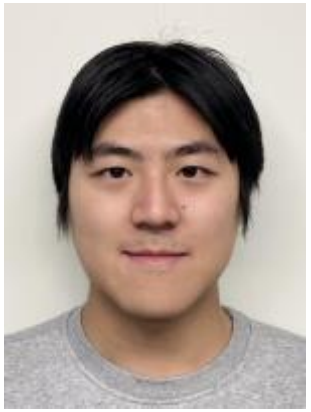
# Introduction to Python for Image Processing

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

# Self-intro



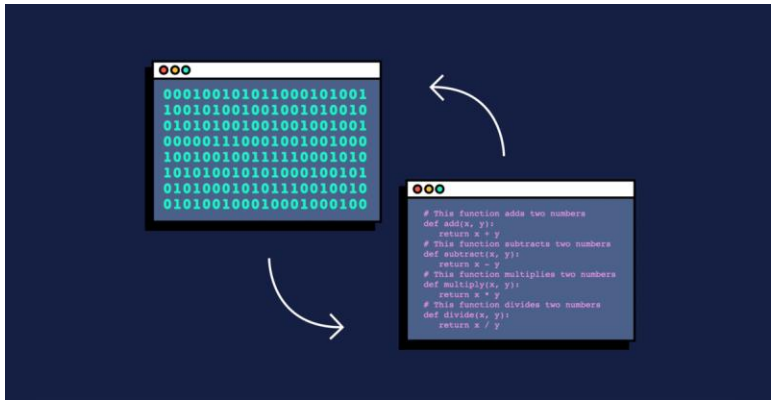
Junyu Chen

- 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!")
```

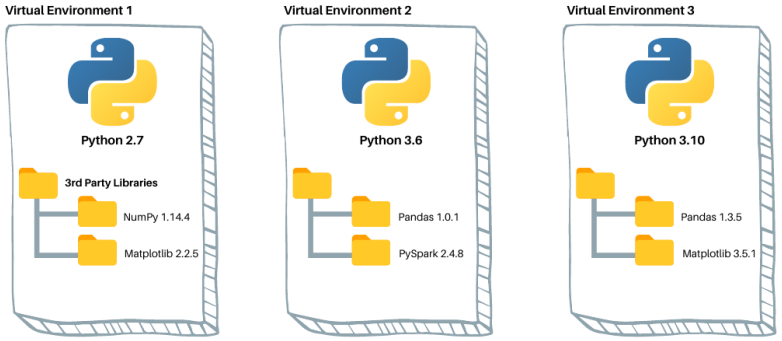
C++:

```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

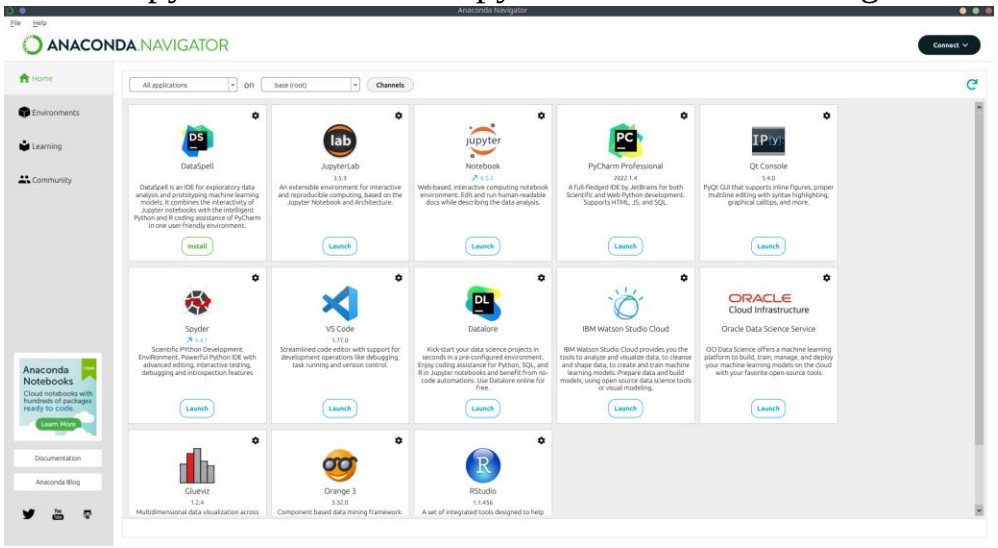



https://www.dataquest.io/blog/a-complete-guide-to-python-virtual-environments/ dataquest.io

- 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

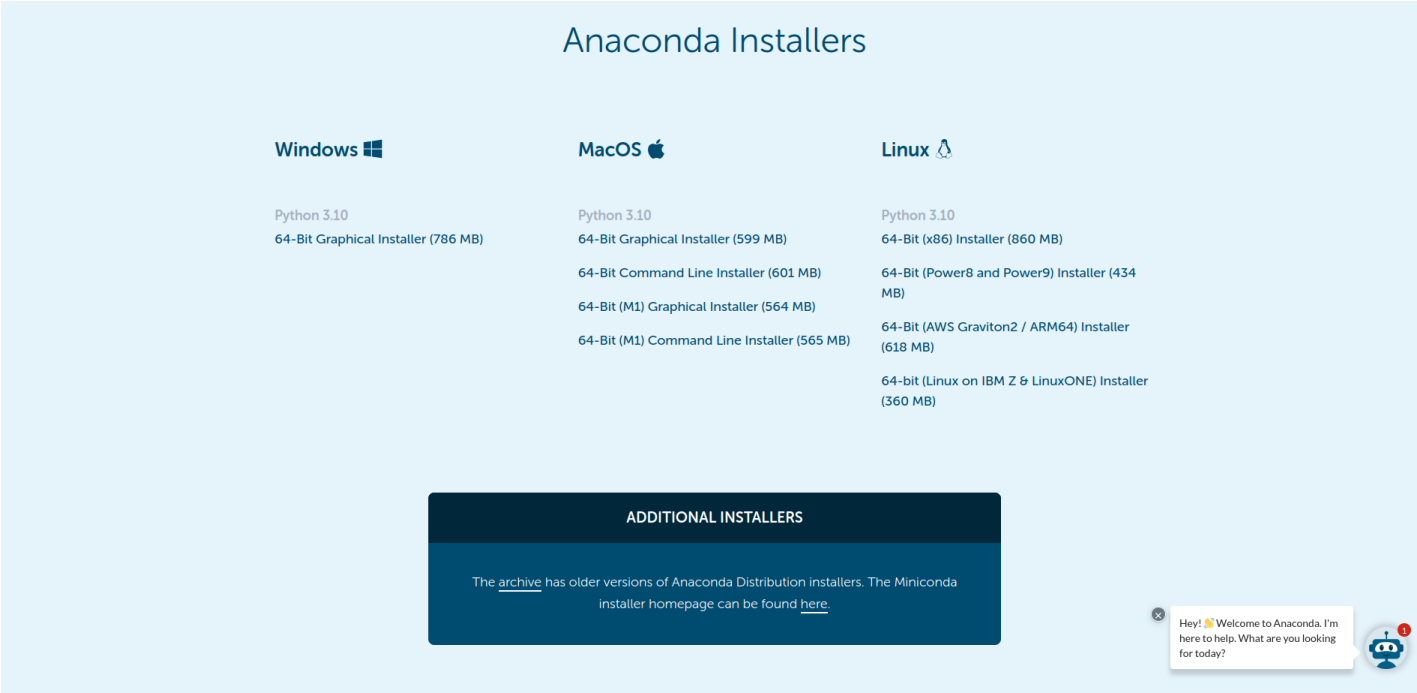
- Anaconda
  - A popular Python distribution that simplifies setting up Python environments, package management, and data science tools.
- Anaconda Navigator
  - A graphical user interface that helps beginners manage environments, packages, and launch applications like Jupyter Notebook and Spyder IDE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



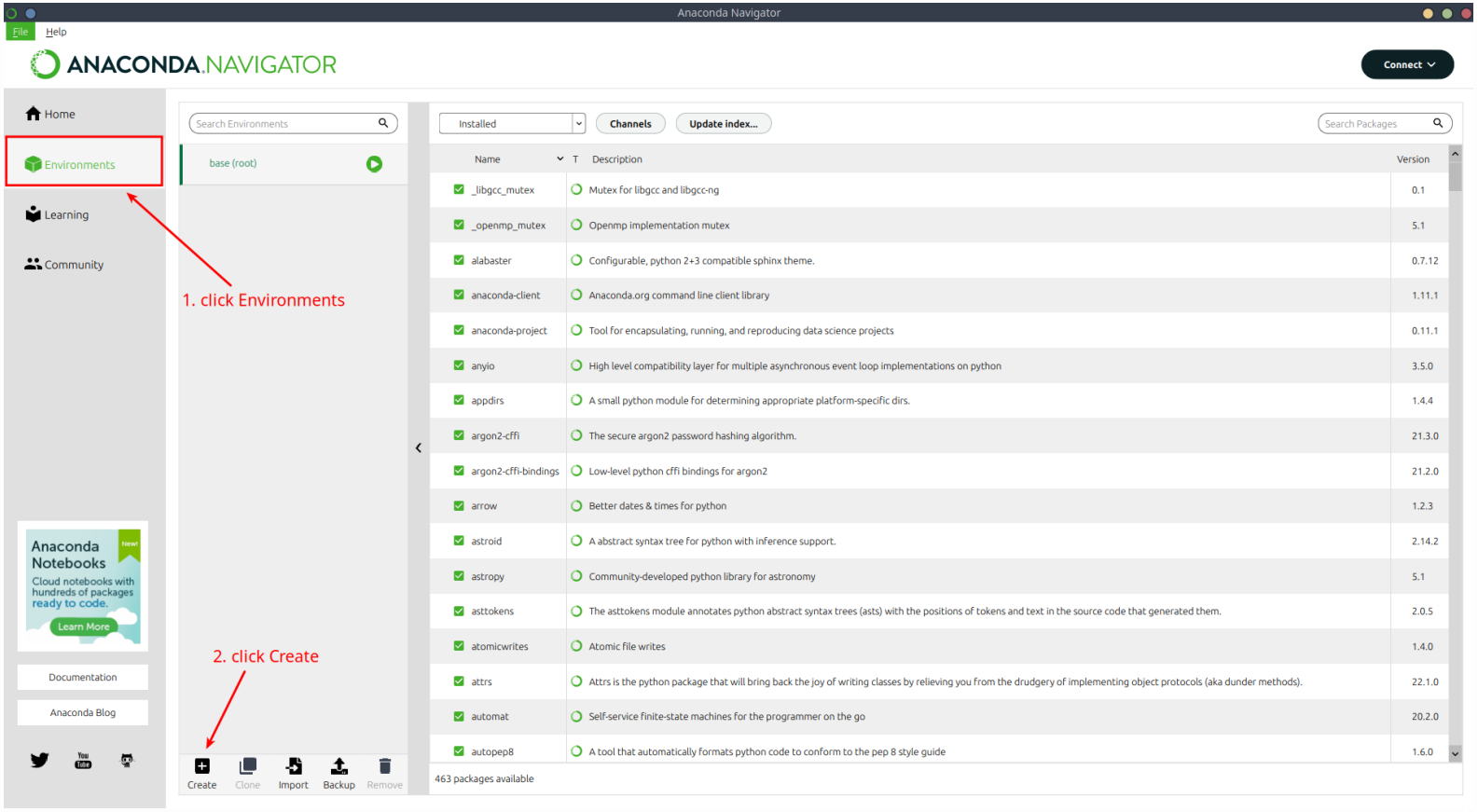

The interface of Anaconda Navigator

# Python Environment with Anaconda

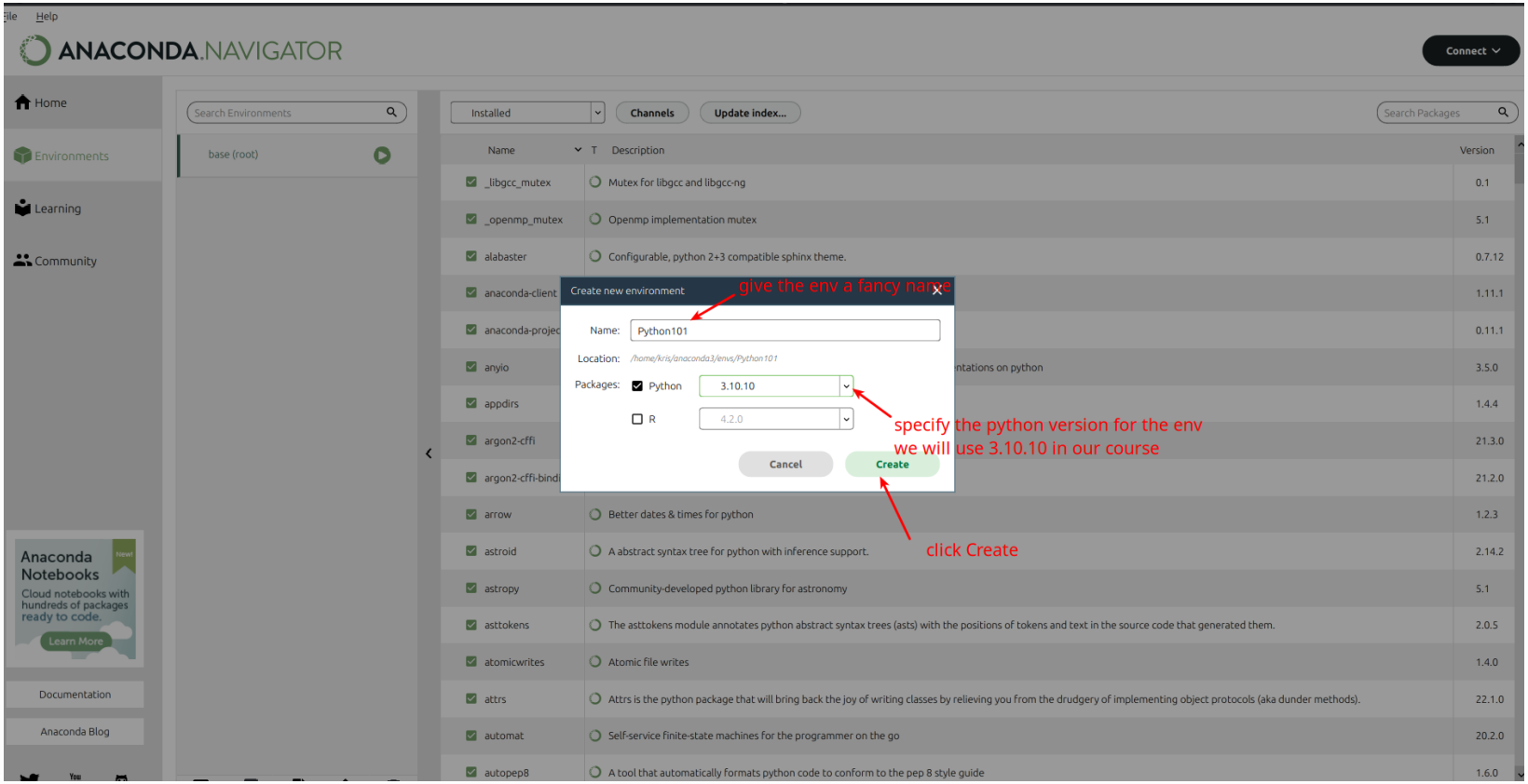
- Install Anaconda Navigator
  - Download link: https://www.anaconda.com/products/distribution#Downloads
  - Tutorial: https://docs.anaconda.com/anaconda/install/



# Python Environment with Anaconda

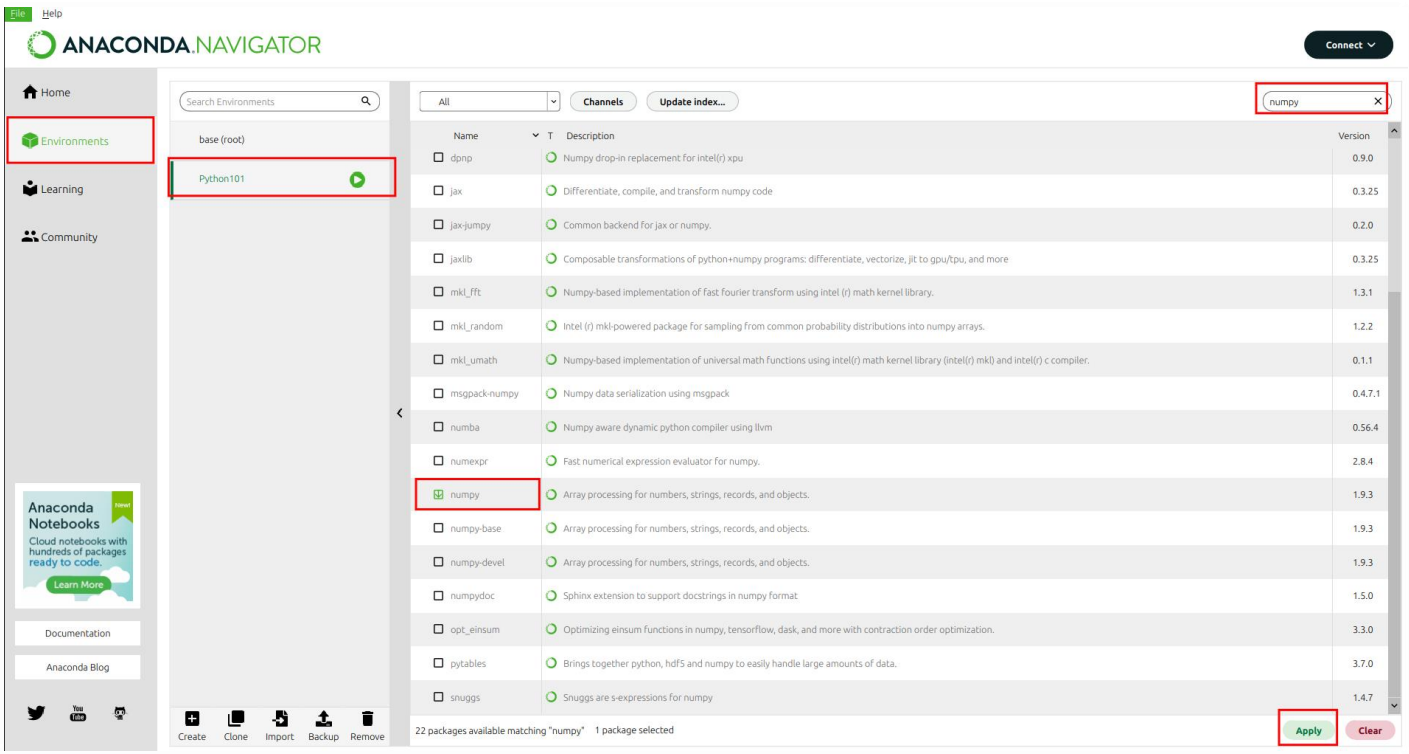


# Python Environment with Anaconda

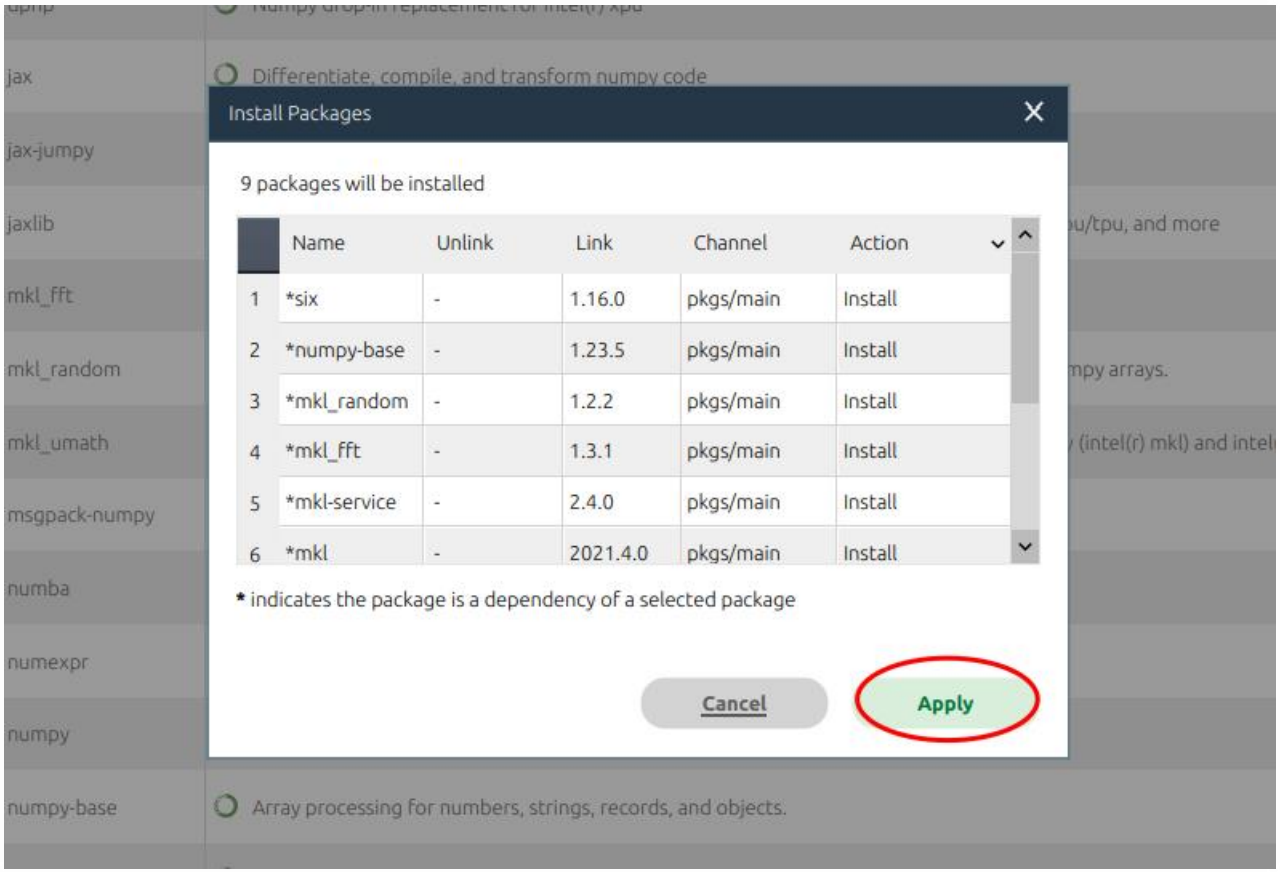


# Python Environment with Anaconda

- Install packages in an environment

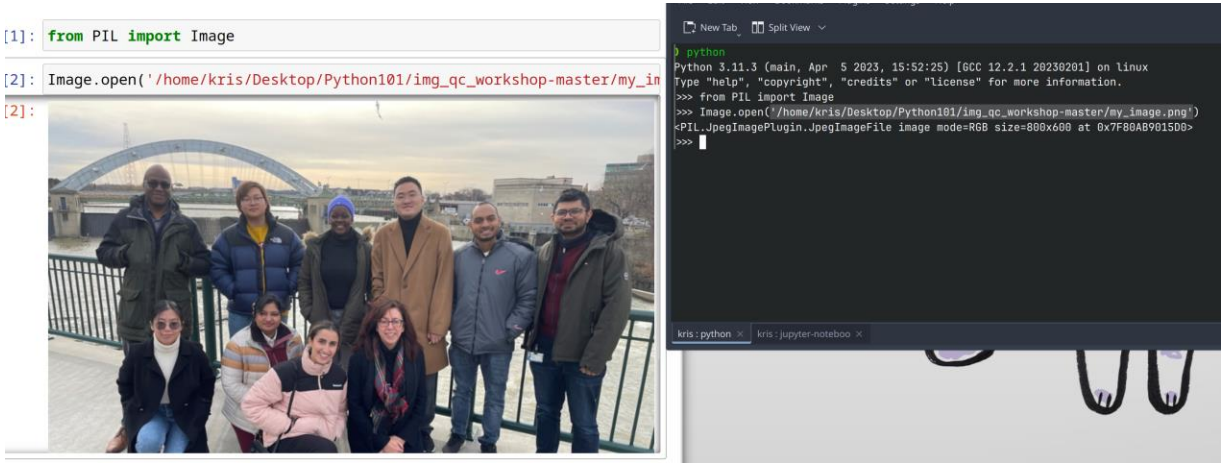


# Python Environment with Anaconda

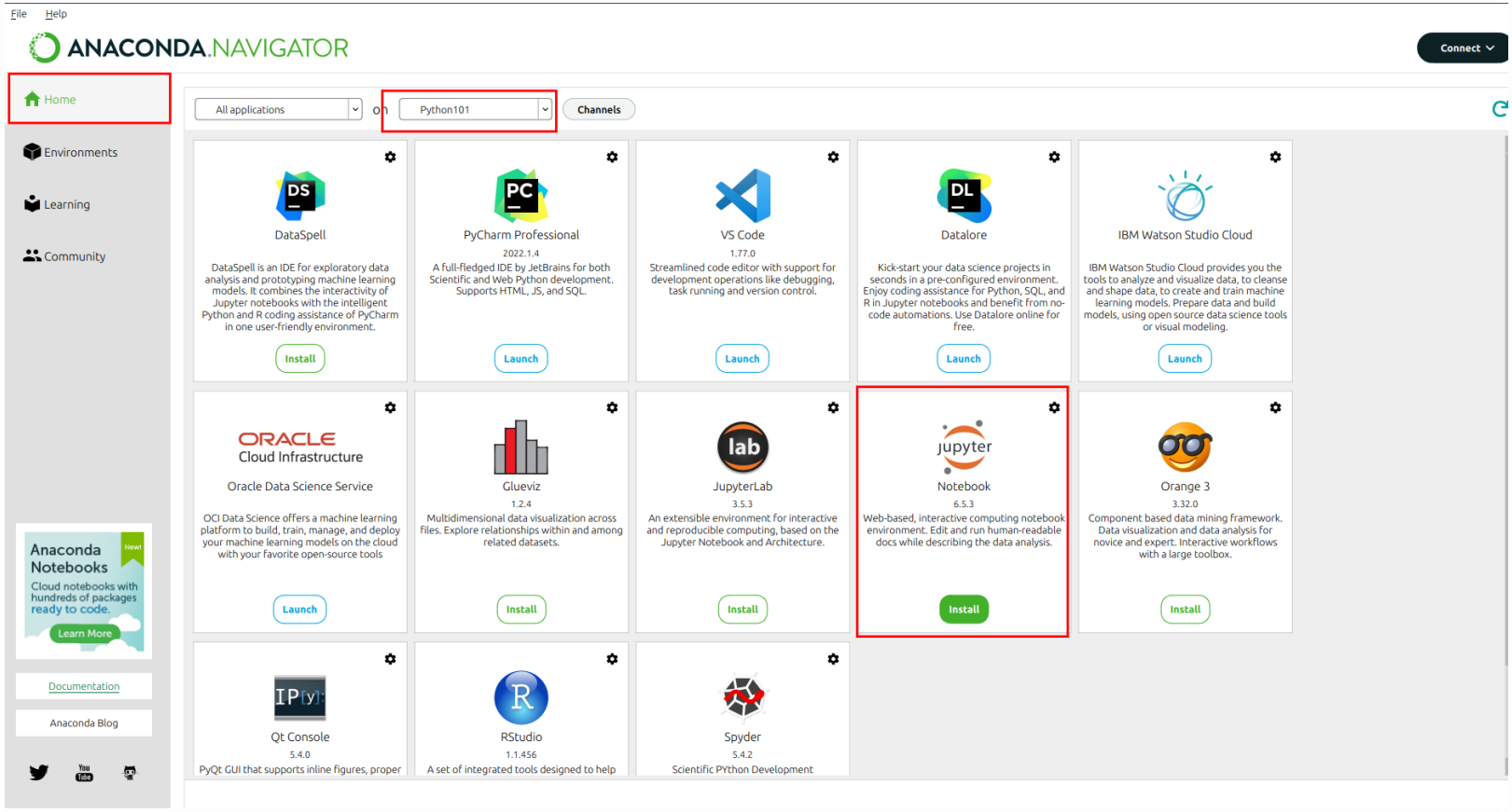


# Python Environment with Anaconda

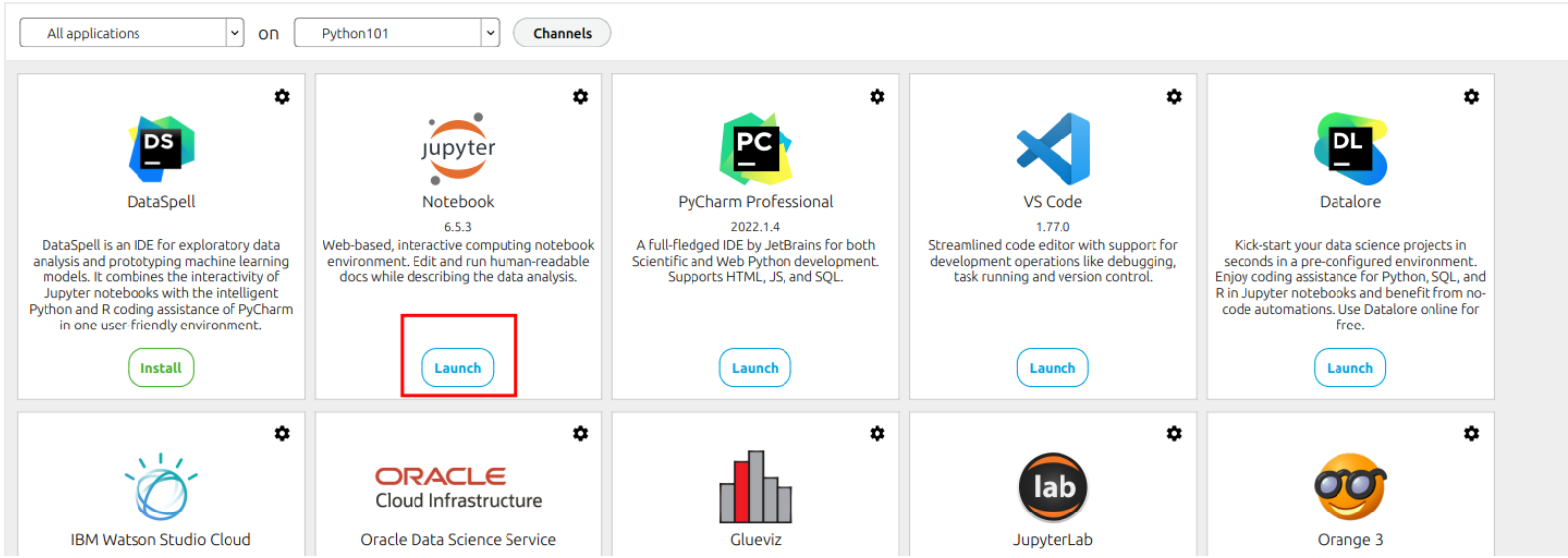
- Jupyter Notebook
  - Interactive coding: write and execute code in an interactive manner
  - Visualization: support the visualization of data using libraries like Matplotlib and Seaborn



# Python Environment with Anaconda



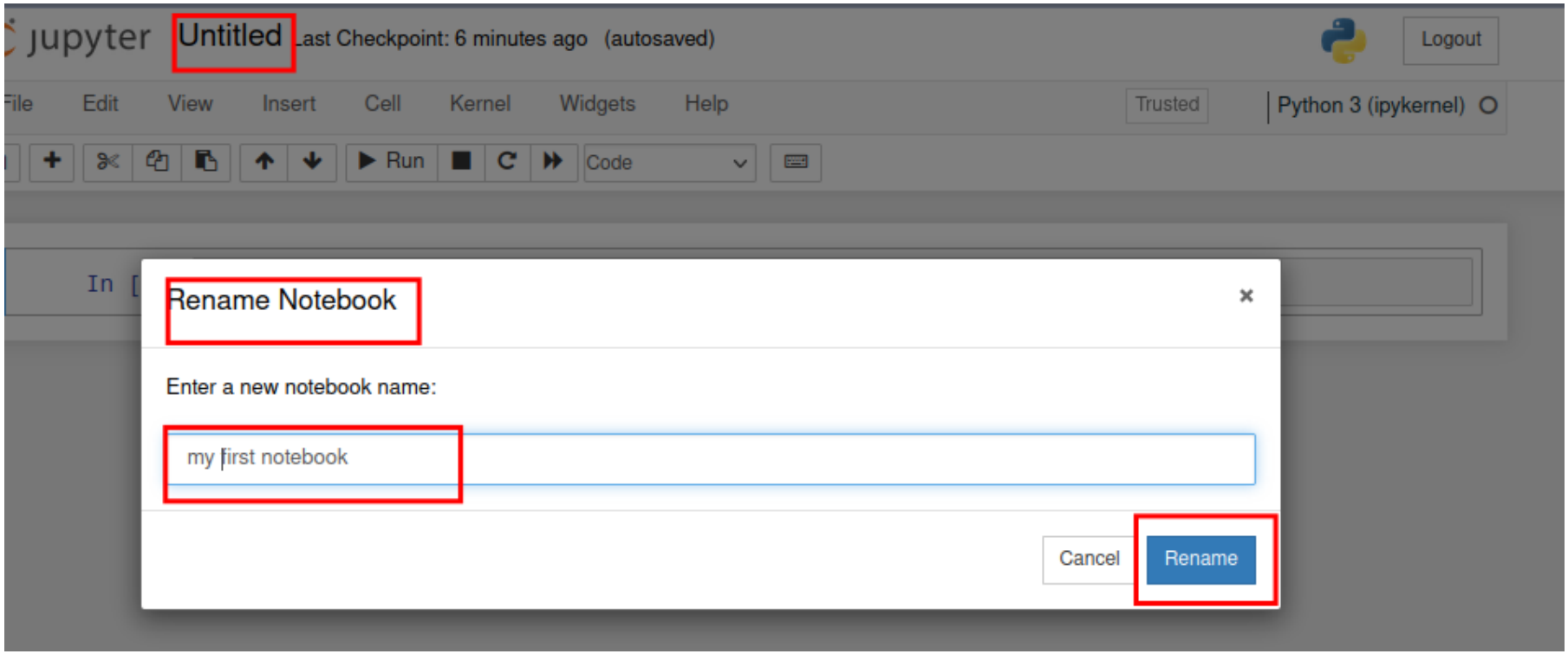
# Python Environment with Anaconda



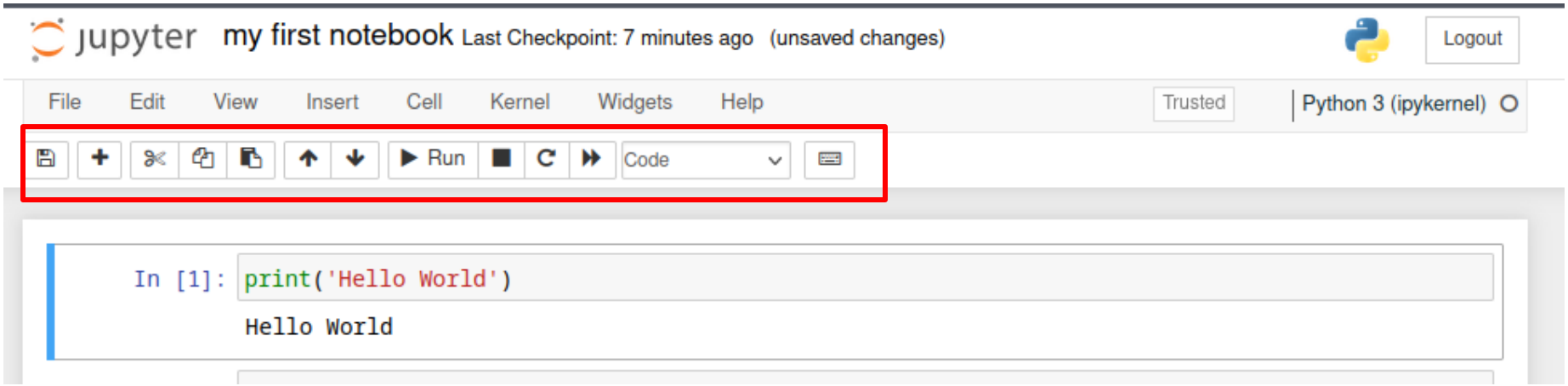
# Python Environment with Anaconda

# Python Environment with Anaconda

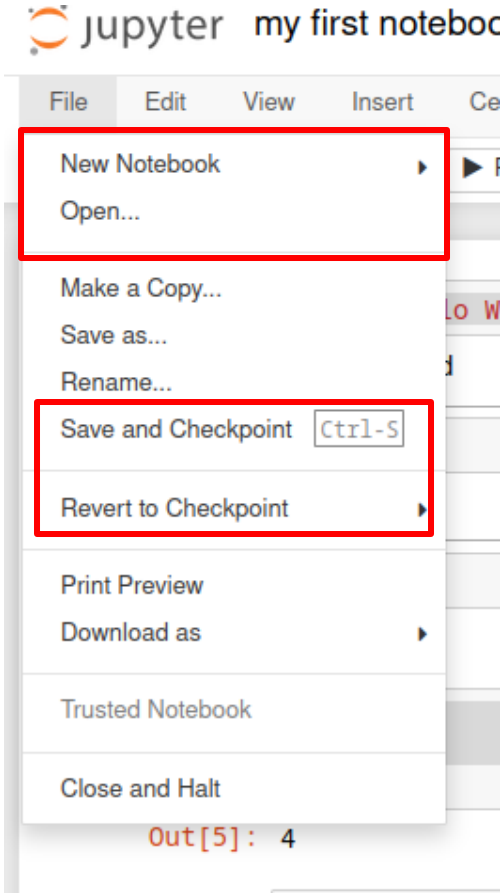
# Python Environment with Anaconda

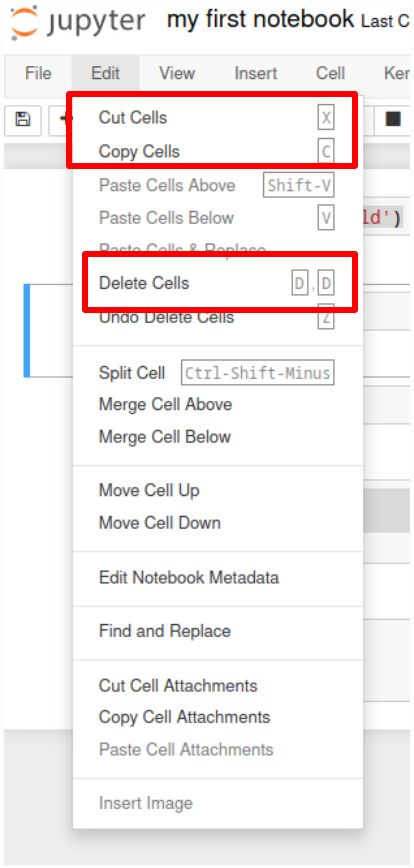


# Python Environment with Anaconda

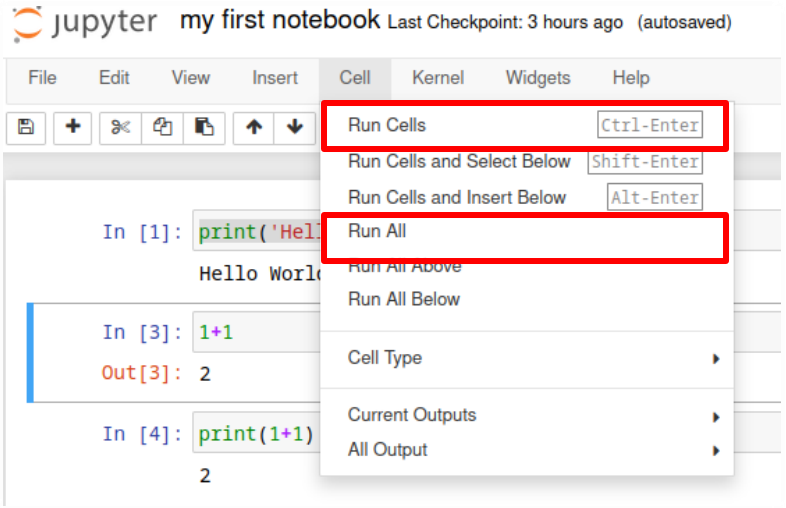


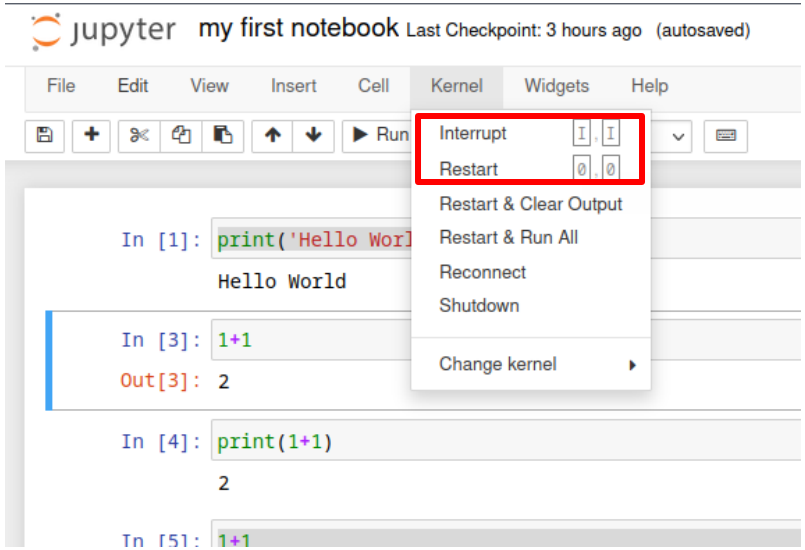
# Python Environment with Anaconda



# 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
```

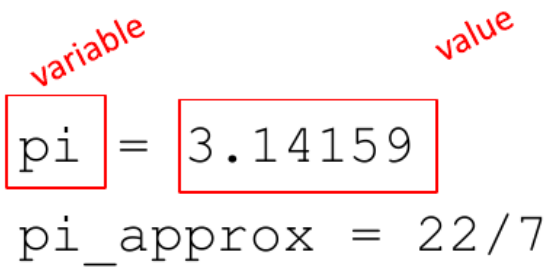
```
1+2*3**4
```

```
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)
```

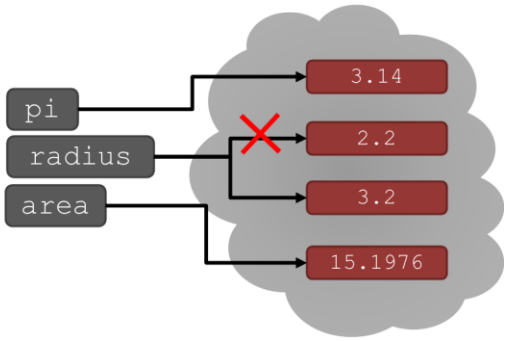
```
2.2
15.205295600000001
```

```
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`

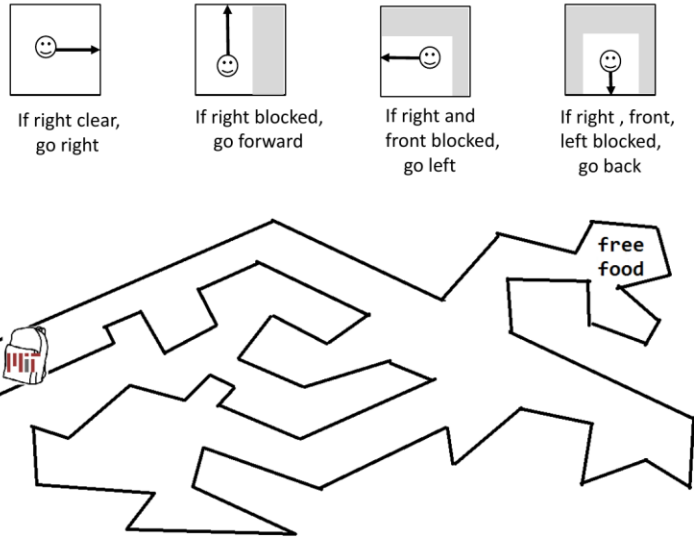
Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# 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

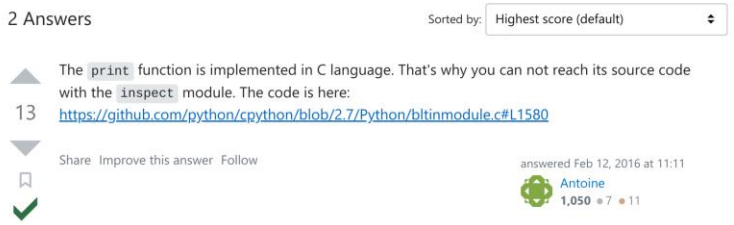
- Write a Python program that prompts the user to enter a word or phrase. The program should then count the number of vowels (a, e, i, o, u) in the input and display the result.
  - Using a **`for`** loop
  - Using a **`while`** loop
- Hint:
  - **`for char in string: # Looping through a string:`**
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

# Functions

- Example – projector
  - a projector is a black box
  - do not know how it works
  - know the interface: input/output
  - connect any electronic to it that can communicate
  - with that input
  - black box somehow converts image from input source
  - to a wall, magnifying it
  - ABSTRACTION IDEA: do not need to know how projector works to use it

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.std
out.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```




https://stackoverflow.com/questions/35360988/how-to-get-source-code-of-python-print-function

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

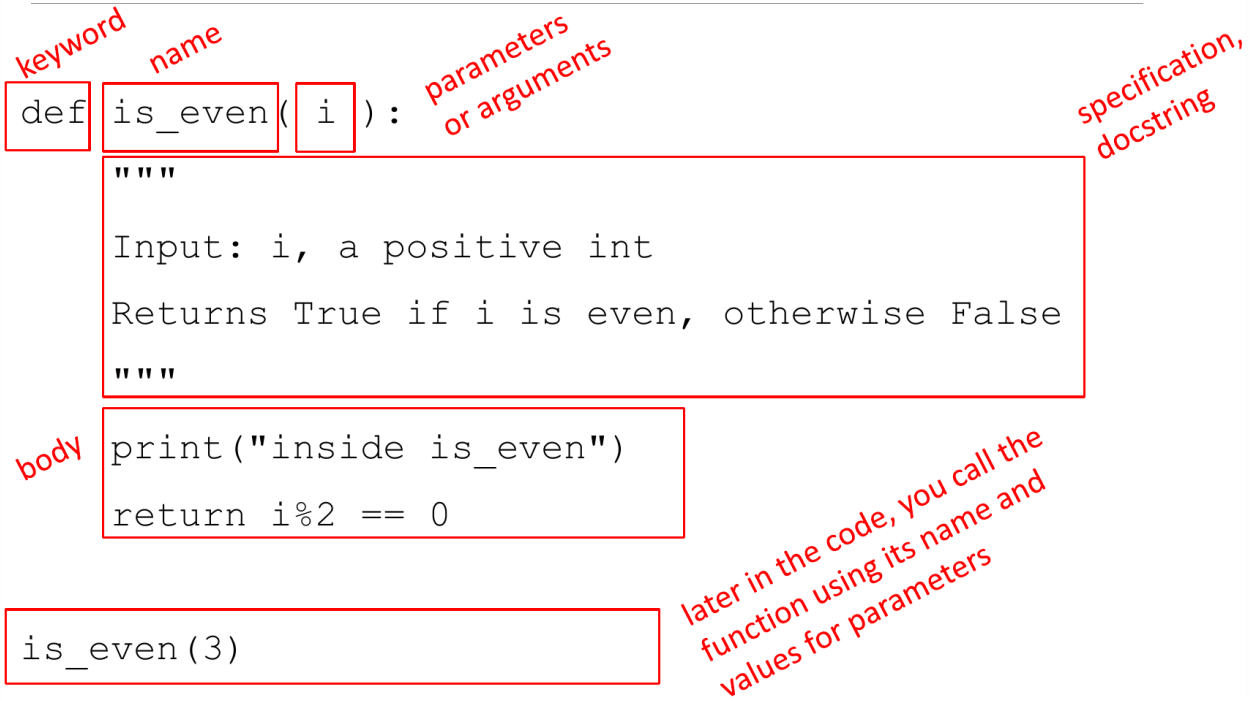
# Functions

- write reusable pieces/chunks of code, called functions
- functions are not run in a program until they are "called" or "invoked" in a program
- function characteristics:
  - has a name
  - has parameters (0 or more)
  - has a docstring (optional but recommended)
  - has a body
  - returns something (0 or more)

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

# Functions

## HOW TO WRITE and CALL/INVOKE A FUNCTION




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

# Functions

## IN THE FUNCTION BODY

```
def is_even( i ):
    """
    Input: i, a positive int
    Returns True if i is even, otherwise False
    """
    print("inside is_even")
    return i%2 == 0
```

run some commands

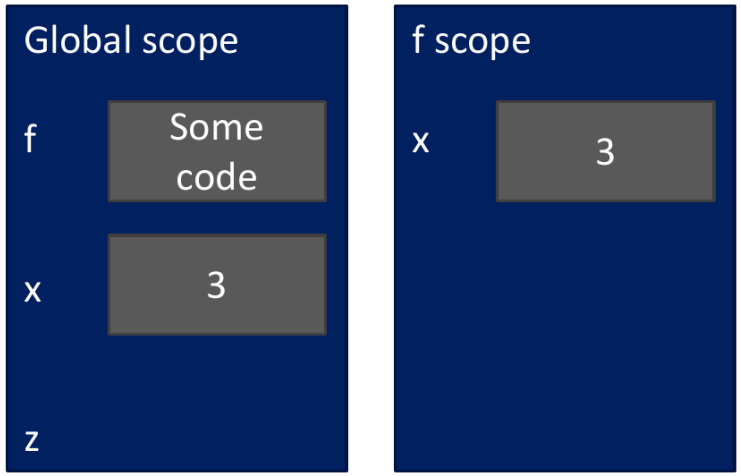
keyword

expression to evaluate and return

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

# Variable Scope

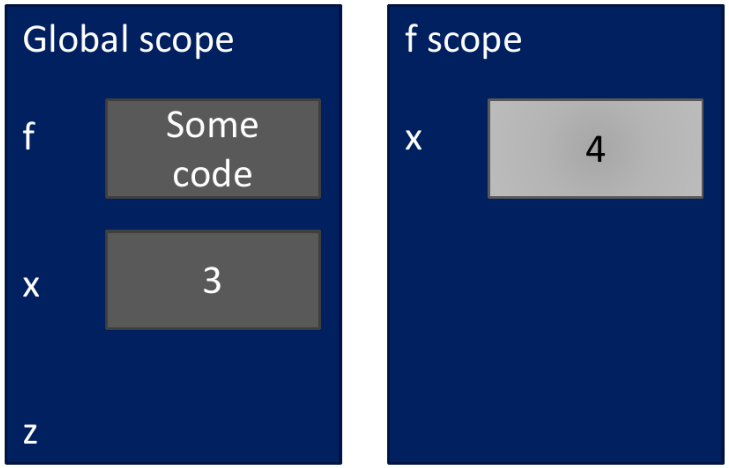
```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```




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

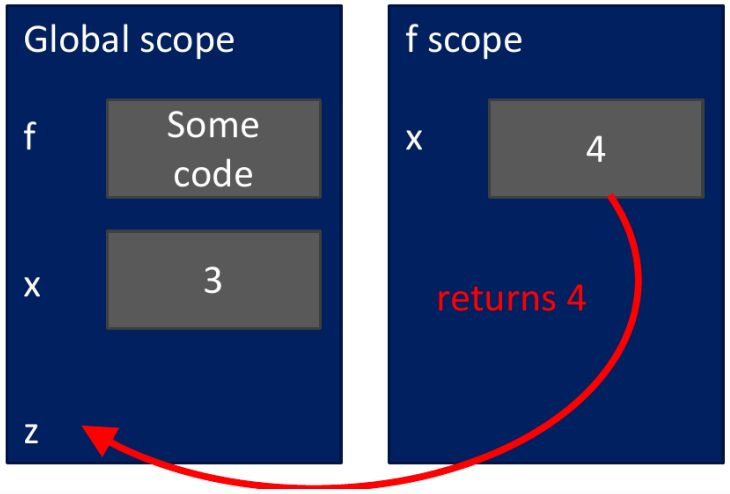
# Variable Scope

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



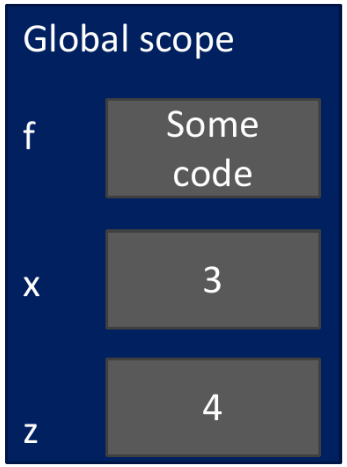
# Variable Scope

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



# Variable Scope

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```




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

# Scope Example

- inside a function, **can access** a variable defined outside
- inside a function, **cannot modify** a variable defined outside -- can using **global variables**, but frowned upon

```
def f(y):
    x = 1
    x += 1
    print(x)

x = 5
f(x)
print(x)
```

x is re-defined in scope of f

different x objects

```
def g(y):
    print(x)
    print(x + 1)

x = 5
g(x)
print(x)
```

x from outside g

x inside g is picked up from scope that called function g

```
def h(y):
    x += 1

x = 5
h(x)
print(x)
```

UnboundLocalError: local variable 'x' referenced before assignment

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

- http://www.pythontutor.com/

# HARDER SCOPE EXAMPLE


***Python Tutor is your best friend to help sort this out!***

***http://www.pythontutor.com/***

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

# If No return is Given:

```
def is_even( i ):
    """
    Input: i, a positive int
    Does not return anything
    """
    i%2 == 0
```

without a return statement

- Python returns the value **None, if no `return` given**
- represents the absence of a value

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

# `return` vs. `print`

- return only has meaning **inside** a function
- only **one** return executed inside a function
- code inside function but after return statement not executed
- has a value associated with it, **given to function caller**

- print can be used **outside** functions
- can execute **many** print statements inside a function
- code inside function can be executed after a print statement
- has a value associated with it, **outputted** to the console

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

# Exercise: Maximum of Two Numbers

- Write a function called **`find_maximum()`** that takes **two** numbers as arguments and returns the maximum of the two numbers. The function should compare the two numbers and determine which one is larger, and then return the larger number as the result.
- **`def find_maximum(num1, num2):`**

# Iteration vs. Recursion

```
def factorial_iter(n):
    prod = 1
    for i in range(1,n+1):
        prod *= i
    return prod
```

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```

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

# Class

- Object-Oriented Programming (OOP):
  - Python is an object-oriented programming language, which means it focuses on creating objects as the primary building blocks of programs.
  - OOP allows us to organize code into modular and reusable units, making it easier to manage and maintain complex projects.

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

# Class

- Class Definition:
  - A class is defined using the **`class`** keyword followed by the name of the class.
  - It serves as a blueprint that defines the properties and behaviors (methods) that objects of that class will possess.
  - For example, a class called **`Car`** can define properties like **`color`**, **`make`**, and **`model`**, as well as behaviors like **`start_engine()`** or **`accelerate()`**.

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

# Class

- Objects and Instances:
  - Once a class is defined, we can create objects or instances of that class.
  - An object is an instance of a class, representing a specific occurrence or entity.
  - For instance, we can create multiple instances of the **`Car`** class, each representing a different car with its own set of properties and behaviors.

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

# Class

In [1]:
```python
# Define a class called 'Person'
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Create an instance of the 'Person' class
person1 = Person("Alice", 25)

# Access attributes of the person1 object
print(person1.name)  # Output: Alice
print(person1.age)   # Output: 25

# Call the greet() method of the person1 object
person1.greet()  # Output: Hello, my name is Alice and I am 25 years old.
```

```
Alice
25
Hello, my name is Alice and I am 25 years old.
```

# Class

In [1]:
```python
# Define a class called 'Person'
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Create an instance of the 'Person' class
person1 = Person("Alice", 25)

# Access attributes of the person1 object
print(person1.name)  # Output: Alice
print(person1.age)   # Output: 25

# Call the greet() method of the person1 object
person1.greet()  # Output: Hello, my name is Alice and I am 25 years old.
```

```
Alice
25
Hello, my name is Alice and I am 25 years old.
```

# Class

In [1]:
```python
# Define a class called 'Person'
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Create an instance of the 'Person' class
person1 = Person("Alice", 25)

# Access attributes of the person1 object
print(person1.name)  # Output: Alice
print(person1.age)   # Output: 25

# Call the greet() method of the person1 object
person1.greet()  # Output: Hello, my name is Alice and I am 25 years old.
```

```
Alice
25
Hello, my name is Alice and I am 25 years old.
```

# Exercise: Bank Account Class

- Define a class called **`BankAccount`**.
- The class should have the following attributes: **`account_number`**, **`owner_name`**, and **`balance`**.
- Implement the **`__init__()`** method to initialize the attributes.
- Implement the following methods:
  - **`deposit(amount)`**: Adds the given amount to the account's balance.
  - **`withdraw(amount)`**: Subtracts the given amount from the account's balance.
  - **`get_balance()`**: Returns the current balance of the account.
  - **`display_account_info()`**: Prints the account information (account number, owner name, and balance).

# Exercise: Bank Account Class

```python
# Define the BankAccount class
class BankAccount:
    def __init__(self, account_number, owner_name, balance=0):
        self... = ...

    def deposit(self, amount):
        # Adds the given amount to the account's balance.

    def withdraw(self, amount):
        # Subtracts the given amount from the account's balance.

    def get_balance(self):
        # Returns the current balance of the account.

    def display_account_info(self):
        # Prints the account information
        # (account number, owner name, and balance).


# Create an instance of the BankAccount class
account1 = BankAccount("123456789", "Alice", 1000)
```

# Tuples

- an ordered sequence of elements, can mix element types
- cannot change element values, **immutable**
- represented with parentheses

remember strings?

```
te = ()
```
empty tuple

```
t = (2,"mit",3)
t[0]                    → evaluates to 2
(2,"mit",3) + (5,6)     → evaluates to (2,"mit",3,5,6)
t[1:2]      → slice tuple, evaluates to ("mit",)
t[1:3]      → slice tuple, evaluates to ("mit",3)
len(t)      → evaluates to 3
t[1] = 4    → gives error, can't modify object
```

extra comma means a tuple with one element

# Tuples

- conveniently used to **swap** variable values

```
x = y
y = x
```

```
temp = x
x = y
y = temp
```

```
(x, y) = (y, x)
```

- used to **return more than one value** from a function

```
def quotient_and_remainder(x, y):
    q = x // y
    r = x % y
    return (q, r)
(quot, rem) = quotient_and_remainder(4,5)
```

integer division

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

# Tuples

- Iterate over tuples

```python
# Define a tuple of fruits
fruits = ('apple', 'banana', 'orange', 'mango')

# Iterate over the elements of the tuple
for fruit in fruits:
    print(fruit)

# Output:
# apple
# banana
# orange
# mango
```

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

# Lists

- **ordered sequence** of information, accessible by index
- a list is denoted by **square brackets**, [ ]
- a list contains **elements**
  - usually homogeneous (ie, all integers)
  - can contain mixed types (not common)
- list elements can be changed so a list is **mutable**

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

# Indices and Ordering

```
a_list = []
```

empty list

```
L = [2, 'a', 4, [1,2]]
len(L)  → evaluates to 4
L[0]    → evaluates to 2
L[2]+1  → evaluates to 5
L[3]    → evaluates to [1,2], another list!
L[4]    → gives an error
i = 2
L[i-1]  → evaluates to 'a' since L[1]='a' above
```

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

# Iterating Over a List

- compute the **sum of elements** of a list
- common pattern, iterate over list elements

like strings, can iterate over list elements directly

```
total = 0
for i in range(len(L)):
    total += L[i]
print total
```

```
total = 0
for i in L:
    total += i
print total
```

- notice
  - list elements are indexed `0` to `len(L)-1`
  - `range(n)` goes from `0` to `n-1`

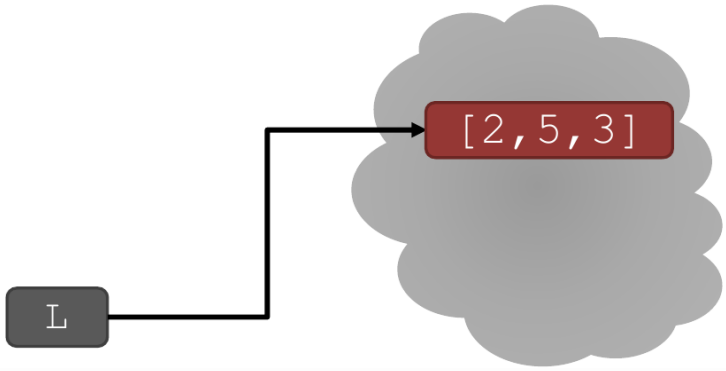
Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Changing Elements

- lists are **mutable**!
- assigning to an element at an index changes the value

```
L = [2, 1, 3]
L[1] = 5
```

- `L` is now `[2, 5, 3]`, note this is the **same object** `L`




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

# Operations on Lists: Add

- **add** elements to end of list with `L.append(element)`
- **mutates** the list!

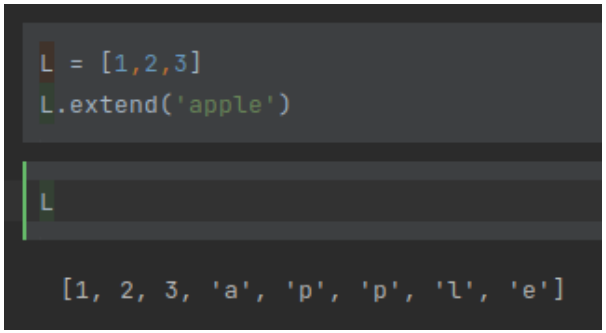
```
L = [2,1,3]
L.append(5)
```

→ L is now `[2,1,3,5]`

- what is the dot?
  - lists are Python objects, everything in Python is an object
  - objects have data
  - objects have methods and functions
  - access this information by `object_name.do_something()`
  - will learn more about these later

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

# Operations on Lists: Add




- to combine lists together use **concatenation**, + operator, to give you a new list
- **mutate** list with `L.extend(some_list)`

```
L1 = [2,1,3]
L2 = [4,5,6]
L3 = L1 + L2
```

→ `L3` is `[2,1,3,4,5,6]`
`L1`, `L2` unchanged

```
L1.extend([0,6])
```

→ mutated `L1` to `[2,1,3,0,6]`

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

# Operations on Lists: Remove

- delete element at a **specific index** with `del(L[index])`
- remove element at **end of list** with `L.pop()`, returns the removed element
- remove a **specific element** with `L.remove(element)`
  - looks for the element and removes it
  - if element occurs multiple times, removes first occurrence
  - if element not in list, gives an error

all these operations mutate the list

```
L = [2,1,3,6,3,7,0] # do below in order
L.remove(2) → mutates L = [1,3,6,3,7,0]
L.remove(3) → mutates L = [1,6,3,7,0]
del(L[1])   → mutates L = [1,3,7,0]
L.pop()     → returns 0 and mutates L = [1,3,7]
```

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

# Exercise: Tuple and List

- Create a tuple named **`my_tuple`** with the following elements: "apple", "banana", "cherry", "apple".
- Create a list named **`my_list`** with the same elements as **`my_tuple`**.
  - Hint: you can use **`list(my_tuple)`**
- Print the length of **`my_tuple`**.
- Print the length of **`my_list`**.
- Add an element "orange" to **`my_list`** using the appropriate method.
- Remove the element "cherry" from **`my_list`** using the appropriate method.
- Print **`my_list`** to verify the modifications.
- Use the **`pop()`** method to remove and print the last element from **`my_list`**.
- Print the modified **`my_list`** to verify the modifications.

# Convert Lists to Srtings

- convert **string to list** with `list(s)`, returns a list with every character from `s` an element in `L`
- can use `s.split()`, to **split a string on a character** parameter, splits on spaces if called without a parameter
- use `''.join(L)` to turn a **list of characters into a string**, can give a character in quotes to add char between every element

```
s = "I<3 cs"          → s is a string
list(s)               → returns ['I','<','3',' ','c','s']
s.split('<')          → returns ['I', '3 cs']
L = ['a','b','c']     → L is a list
''.join(L)            → returns "abc"
'_'.join(L)           → returns "a_b_c"
```

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

# Other List Operations

- `sort()` and `sorted()`
- `reverse()`
- and many more!
  https://docs.python.org/3/tutorial/datastructures.html

```
L=[9,6,0,3]
```

`sorted(L)` → returns sorted list, does **not mutate** `L`

`L.sort()` → **mutates** `L=[0,3,6,9]`

`L.reverse()` → **mutates** `L=[9,6,3,0]`

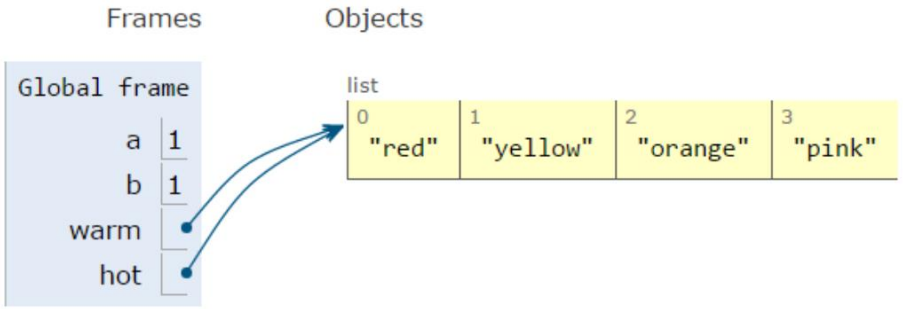
Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Aliases

- `hot` is an **alias** for `warm` – changing one changes the other!
- `append()` has a side effect

```
1  a = 1
2  b = a
3  print(a)
4  print(b)
5
6  warm = ['red', 'yellow', 'orange']
7  hot = warm
8  hot.append('pink')
9  print(hot)
10 print(warm)
```

```
1
1
['red', 'yellow', 'orange', 'pink']
['red', 'yellow', 'orange', 'pink']
```




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

# Cloning a List

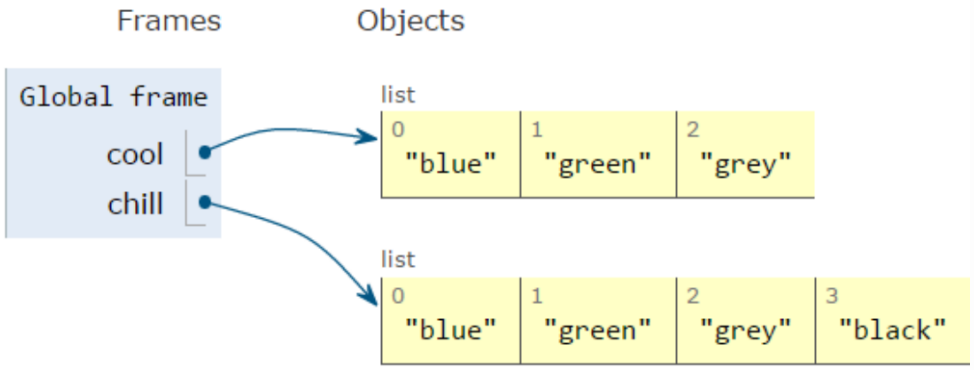
```
L_1 = [1,2,3]
L_2 = L_1.copy()
L_1.append(4)
print(L_1)
print(L_2)
```

```
[1, 2, 3, 4]
[1, 2, 3]
```

- create a new list and **copy every element** using `chill = cool[:]`

```
1 cool = ['blue', 'green', 'grey']
2 chill = cool[:]
3 chill.append('black')
4 print(chill)
5 print(cool)
```

```
['blue', 'green', 'grey', 'black']
['blue', 'green', 'grey']
```



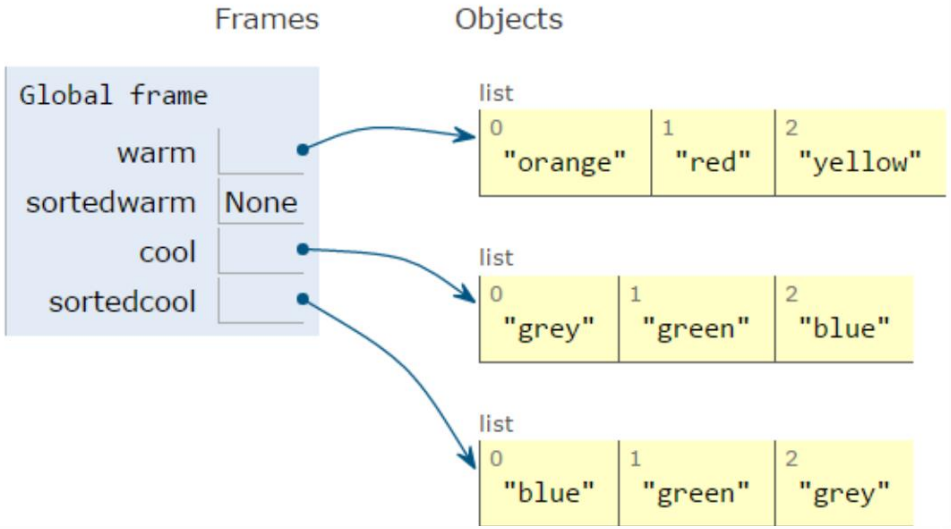

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

# Cloning a List

- calling `sort()` **mutates** the list, returns nothing
- calling `sorted()` **does not mutate** list, must assign result to a variable

```
['orange', 'red', 'yellow']
None
['grey', 'green', 'blue']
['blue', 'green', 'grey']
```

```
1 warm = ['red', 'yellow', 'orange']
2 sortedwarm = warm.sort()
3 print(warm)
4 print(sortedwarm)
5
6 cool = ['grey', 'green', 'blue']
7 sortedcool = sorted(cool)
8 print(cool)
9 print(sortedcool)
```

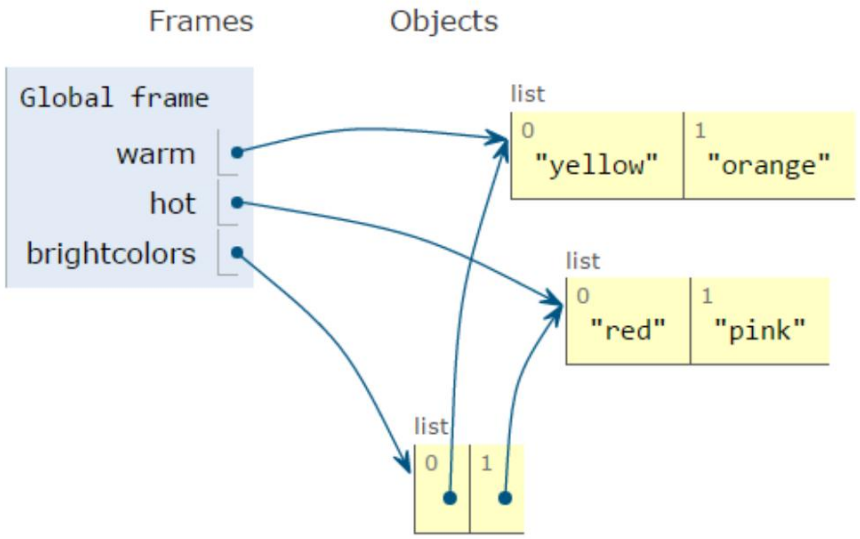


# Nested Lists

- can have **nested** lists
- side effects still possible after mutation

```
[['yellow', 'orange'], ['red']]
['red', 'pink']
[['yellow', 'orange'], ['red', 'pink']]
```

```
1 warm = ['yellow', 'orange']
2 hot = ['red']
3 brightcolors = [warm]
4 brightcolors.append(hot)
5 print(brightcolors)
6 hot.append('pink')
7 print(hot)
8 print(brightcolors)
```




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

# Avoiding mutation during iteration

- **avoid** mutating a list as you are iterating over it

```
def remove_dups(L1, L2):
    for e in L1:
        if e in L2:
            L1.remove(e)
```

```
def remove_dups(L1, L2):
    L1_copy = L1[:]
    for e in L1_copy:
        if e in L2:
            L1.remove(e)
```

```
L1 = [1, 2, 3, 4]
L2 = [1, 2, 5, 6]
remove_dups(L1, L2)
```

clone list first, note that `L1_copy = L1` does NOT clone

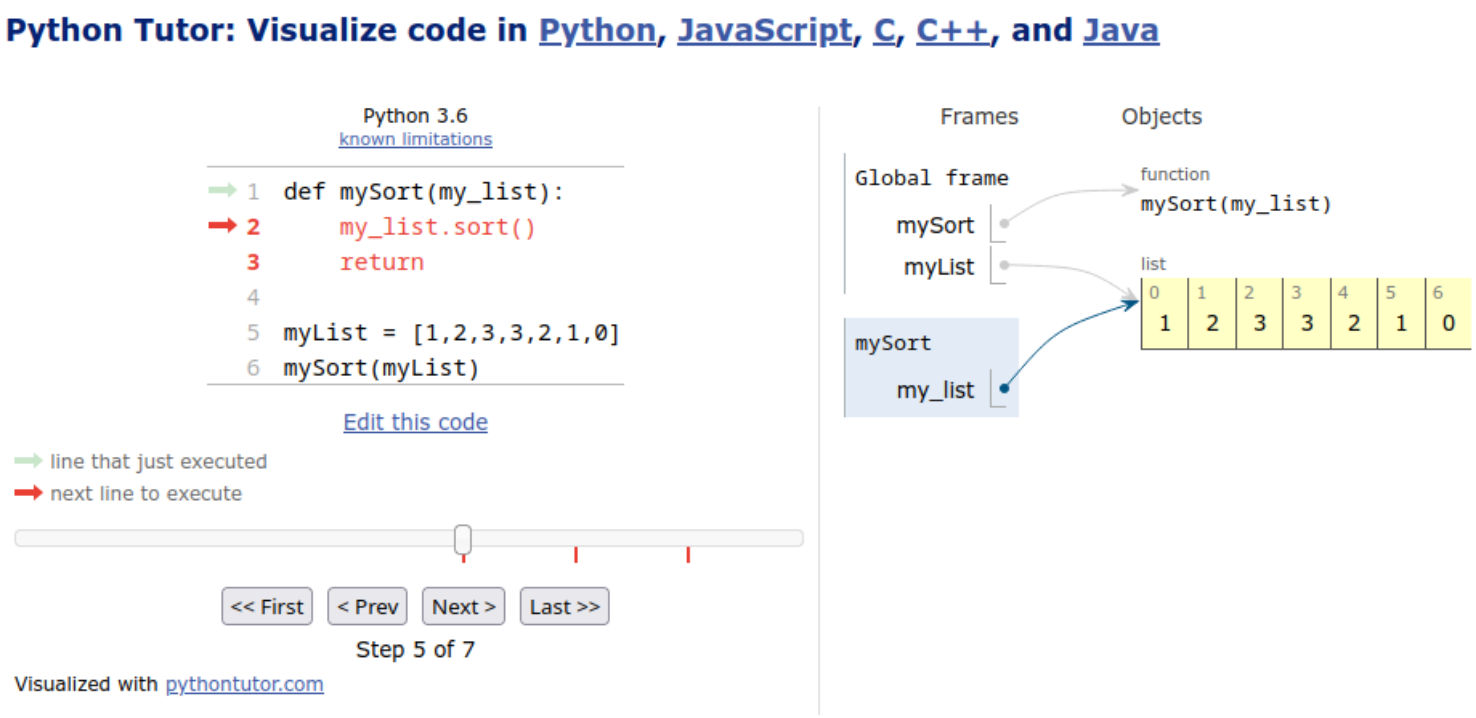
- `L1` is `[2,3,4]` not `[3,4]` Why?
  - Python uses an internal counter to keep track of index it is in the loop
  - mutating changes the list length but Python doesn't update the counter
  - loop never sees element 2

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

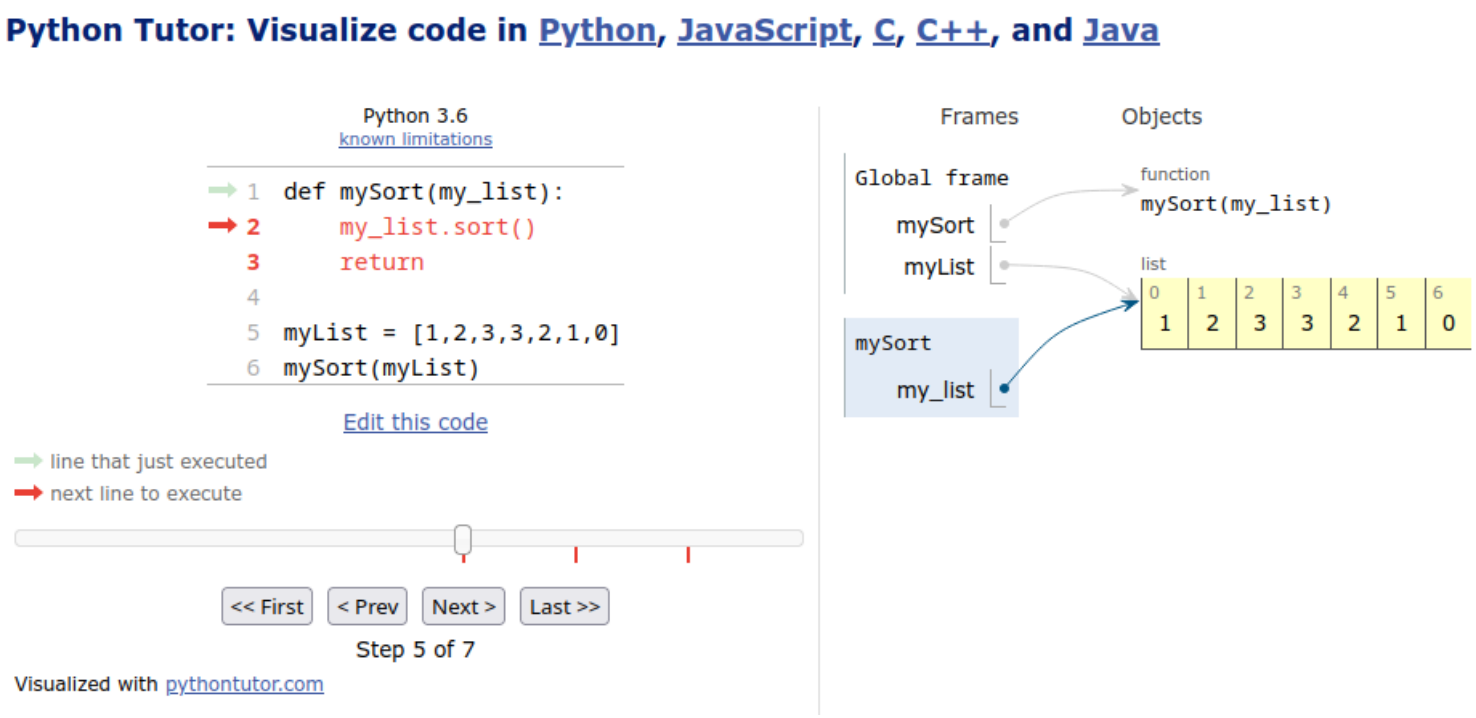
# Exercise: List Operations Function

1. Write a function called **`list_operations`** that takes a list as an argument.
2. Within the function, perform the following operations on the list:
   - **Clone the original list** and assign it to a new variable.
   - Sort the original list in ascending order.
   - **Create an alias of the original list** by assigning it to a new variable.
   - Reverse the alias of the list
3. Return the cloned list, sorted list, and the alias of the original list.
4. Test the function by calling it with different lists and printing the results.

# Exercise: List Operations Function



# Exercise: List Operations Function



- Dictionary
  - e.g., how to store student info

    - so far, can store using separate lists for every info

```
names = ['Ana', 'John', 'Denise', 'Katy']
grade = ['B', 'A+', 'A', 'A']
course = [2.00, 6.0001, 20.002, 9.01]
```

    - a **separate list** for each item
    - each list must have the **same length**
    - info stored across lists at **same index**, each index refers to info for a different person

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

- Dictionary
  - e.g., how to store student info

```
def get_grade(student, name_list, grade_list, course_list):
    i = name_list.index(student)
    grade = grade_list[i]
    course = course_list[i]
    return (course, grade)
```

- **messy** if have a lot of different info to keep track of
- must maintain **many lists** and pass them as arguments
- must **always index** using integers
- must remember to change multiple lists

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

- Dictionary
  - e.g., how to change student info
  - dict: a better and clearner way
  - nice to **index item of interest directly** (not always int)
  - nice to use **one data structure**, no separate lists

**A list**

| | |
|---|---|
| 0 | Elem 1 |
| 1 | Elem 2 |
| 2 | Elem 3 |
| 3 | Elem 4 |
| … | … |

index element

**A dictionary**

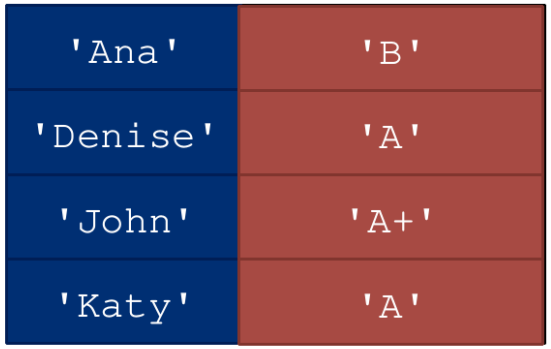
| | |
|---|---|
| Key 1 | Val 1 |
| Key 2 | Val 2 |
| Key 3 | Val 3 |
| Key 4 | Val 4 |
| … | … |

custom index by label element

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

# Python Dictionary

- store pairs of data
  - key
  - value




empty dictionary

```
my_dict = {}
```

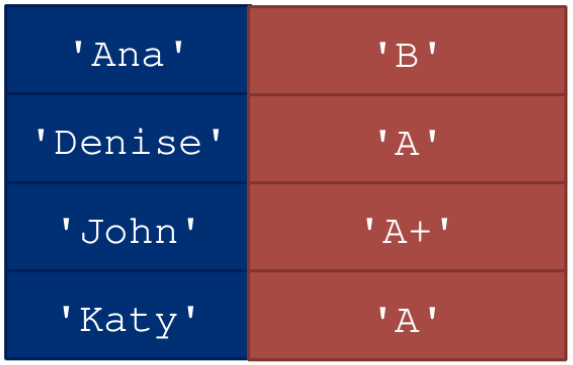
```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

key1 val1 key2 val2 key3 val3 key4 val4

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

# Dictionary Lookup

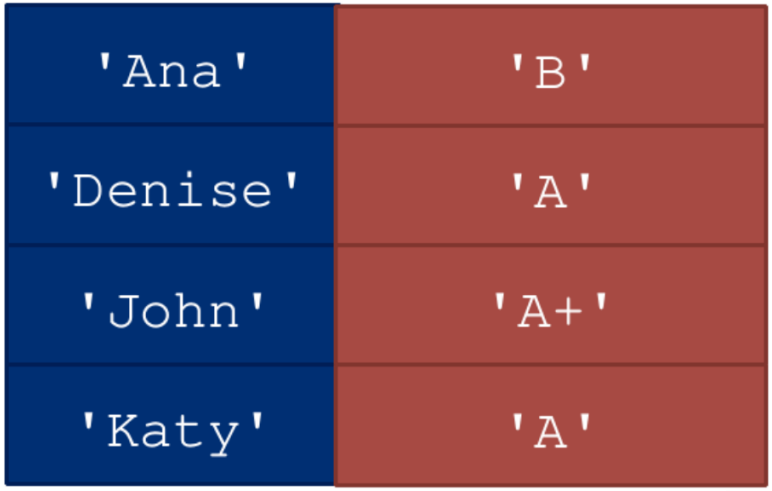
- similar to indexing into a list
- **looks up** the **key**
- **returns** the **value** associated with the key
- if key isn't found, get an error




```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
grades['John']      → evaluates to 'A+'
grades['Sylvan']    → gives a KeyError
```

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

# Dictionary Operations




```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

- **add** an entry

```
grades['Sylvan'] = 'A'
```

- **test** if key in dictionary

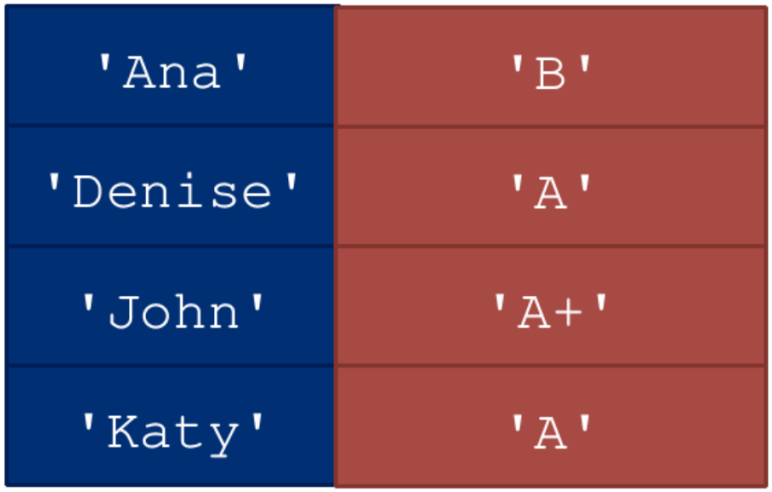
```
'John' in grades        → returns True
'Daniel' in grades      → returns False
```

- **delete** entry

```
del(grades['Ana'])
```

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

# Dictionary Operations




```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

- get an **iterable that acts like a tuple of all keys** (no guaranteed order)

  `grades.keys()` → returns `['Denise','Katy','John','Ana']`

- get an **iterable that acts like a tuple of all values**

  `grades.values()` → returns `['A', 'A', 'A+', 'B']` (no guaranteed order)

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

# Dictionary Keys and Values

```python
students = {
    "John": {"grade": 85, "course": "Math"},
    "Jane": {"grade": 92, "course": "Science"},
    "Michael": {"grade": 78, "course": "English"}
}
```

- values
  - any type (**immutable and mutable**)
  - can be **duplicates**
  - dictionary values can be lists, even other dictionaries!
- keys
  - must be **unique**
  - **immutable** type (`int`, `float`, `string`, `tuple`, `bool`)
    - actually need an object that is **hashable,** but think of as immutable as all immutable types are hashable
  - careful with `float` type as a key
- **no order** to keys or values!

```
d = {4:{1:0}, (1,3):"twelve", 'const':[3.14,2.7,8.44]}
```

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

# `list` vs `dict`

- **ordered** sequence of elements
- look up elements by an integer index
- indices have an **order**
- index is an **integer**

- **matches** "keys" to "values"
- look up one item by another item
- **no order** is guaranteed
- key can be any **immutable** type

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

# Exercise: Student Database

- Create an empty dictionary called **`student_db`**.
- Add the following student records to the **`student_db`** dictionary; using student id as the key:
  - Student ID: 101, Name: Alice, Grade: A
  - Student ID: 102, Name: Bob, Grade: B
  - Student ID: 103, Name: Charlie, Grade: C

```
student_db = {}
student_db[some_id] = {"Name": "some_name", "Grade": "some_grade"}
```

- Print the student records in the following format:
- Update the grade of student with ID 102 to "A+".
- Print the updated student record of student with ID 102.
- Check if student with ID 104 exists in the **`student_db`** dictionary.
- Print "Student ID 104 exists" if it exists, or "Student ID 104 does not exist" if it doesn't.
- Remove the student record of student with ID 103 from the **`student_db`** dictionary.
- Print the remaining student records after the removal in the same format as step 3.
- Clear all records from the **`student_db`** dictionary.
- Print the dictionary to confirm that it is empty.