The UNIVERSITY of OKLAHOMA

Coding Outreach for Data Education (CODE) Workshop

Workshop

Introduction to Python (Part 1)

- Instructor: Nasrin Akter
- Teaching Assistant: Manisha Mandava

Nasrin Akter

Website: https://sites.create.ou.edu/nasrinakter/
Linkedin: https://www.linkedin.com/in/nasrin-akter-ece/

- Ph.D. student in Electrical and Computer Engineering, The University of Oklahoma | Norman, Oklahoma.
- MS in Computer Science and Engineering, Dhaka, Bangladesh.
- BSc in Electronics and Telecommunication Engineering, Dhaka, Bangladesh.
- Graduate teaching assistant.
- Machine Learning Research Intern at Hearts for Hearing, Oklahoma City.
- Corporate Outreach Chair, Society of Women Engineers
- 5 years Python experience, 8+ years of Industrial experience.
- Current research: Machine Learning, Digital Image Processing, Computer Vision, Medical Image Informatics.

My skills in brief include:

• Data Visualization & Analytics: Tableau, Microsoft Power BI.
• Programming Language: Python, C/C++, MATLAB.
• Deep Learning Frameworks: Keras, TensorFlow.
• DBMS: MySql.
• IDEs: Jupyter Notebook, Google Colab,
• Deep Learning algorithm: ANN, CNN, RNN
• Transmission/ Telephony Technology: SDH Multiplexing, SS7 Signaling, Voice Codecs, H.323.
• Networking: HTTP, OSI model, TCP/IP, Routing protocols, IPv6 addressing and address Planning.
• Network management tools and simulation software: packet analyzer, Cacti.

Publications

- Apriori-backed Fuzzy Unification and Statistical Inference in Feature Reduction | **Springer**
- Prediction of Academic Performance Applying NN | **IJACSA**
- Detection of Autism Spectrum Disorder applying Deep Neural Network

Projects

- EEG data-driven Machine Learning for classification of stroke and healthy subjects
- Classification the Gait cycle images to detect diabetic neuropathy among cancer patients
- Automatic Brain Tumor segmentation on MRI images using Deep Learning
- Automatic Breast Tumor Segmentation And Classification: A Deep Learning Approach

Teaching Assistant

Manisha Mandava

- M.S. in Computer Science
  (OU, 2021 - 2023)
- Graduate teaching assistant

Schedule

Day 1

- Install Python and open IDLE
- Interactive and batch mode
- Docstrings and comments
- Learn the basics of Python syntax
  - Variables
  - Data types
  - Strings

Day 2

- Review the basics of Python syntax
- Practice Python in IDLE
- In class exercise
- Lists
- Loops

You don't need to know what any of these mean yet!

Introduction

Course Objectives

- Give scientists comfort and confidence with computation
- Learn and practice computational skills for your future or current research
- Basic Python programming
  - Learn how to use IDLE to execute Python commands
  - Learn the basics of Python syntax
  - Explore Python data types, create and manipulate variables, and *use loops in a Python program*

Introduction

- **What you need to succeed:**
  - A logical and organized way of thinking (or at least organized notes for your future self!)
  - Determination
  - Practice, practice, practice
- **What you don't need:**
  - A math background
  - Previous programming experience

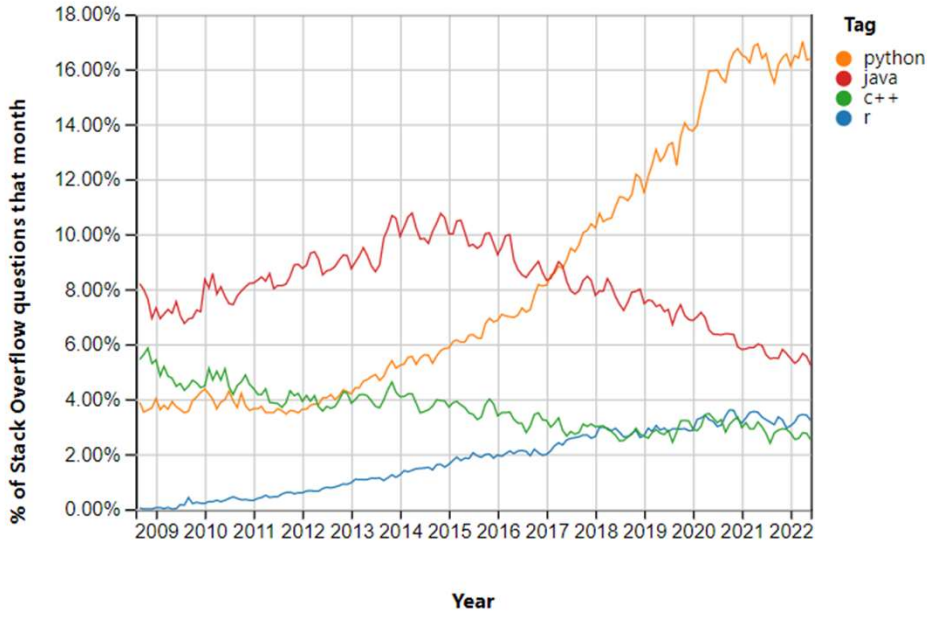
Let’s get started!!

What is Programming?

- Providing a set of instructions for a computer to execute
- A way to automate tasks, perform difficult data analysis, document the process, and save time
- There are many programming languages and applications
  - Python
  - Java
  - R
  - Perl

Why Python?

- Easy to learn
- Popular
- Free and open-source
- Interchangeable between Windows, Linux, and iOS
- High-level and object-oriented
- Many applications use Python



What is a Program(Script)?

- A detailed set of instructions for how to do something
  - The code calculates the area of circles with radii 1 and 2.
  - It uses the value of π (pi) as approximately 3.1416.
  - An empty list called area is created to store the calculated areas.
  - The for loop runs twice, with r taking the values 1 and 2.
  - For each r, the area is calculated using the formula: Area = π * r ** 2.
  - The calculated area is added to the area list using the append() method.
  - After the loop, the area list contains the areas of the circles with radii 1 and 2: [3.1416, 12.5664].

```
In [56]: # Definitions/symbols
         pi = 3.1416

         # Initialize an empty list
         area=[]

         for r in range(1, 3):
             Area=pi*r**2
             area.append(Area)
```

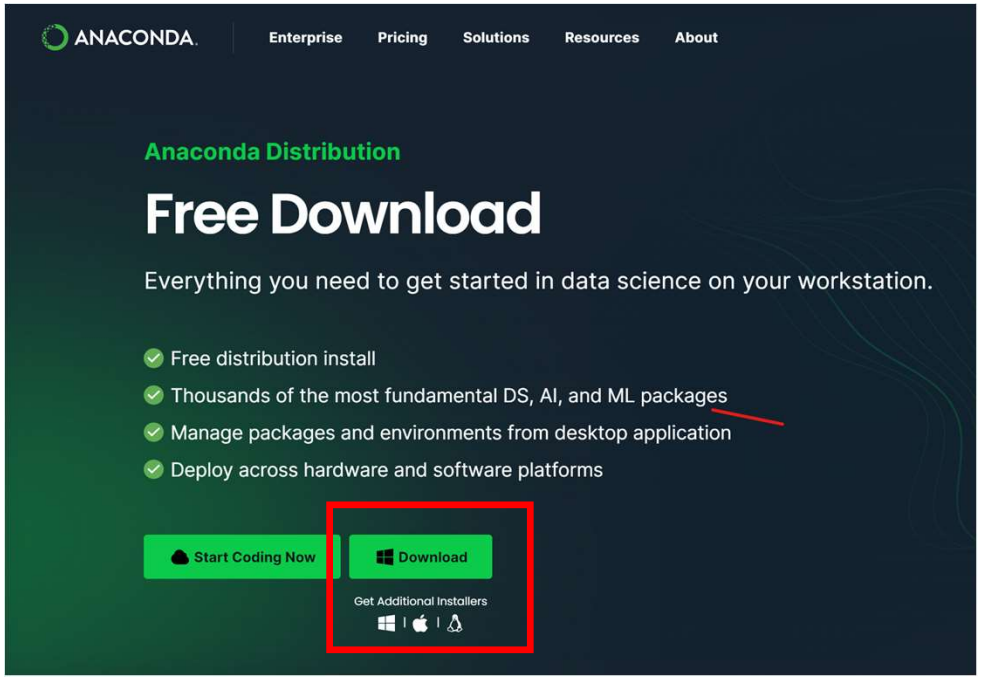
```
In [57]: area

Out[57]: [3.1416, 12.5664]
```

Setup Environment

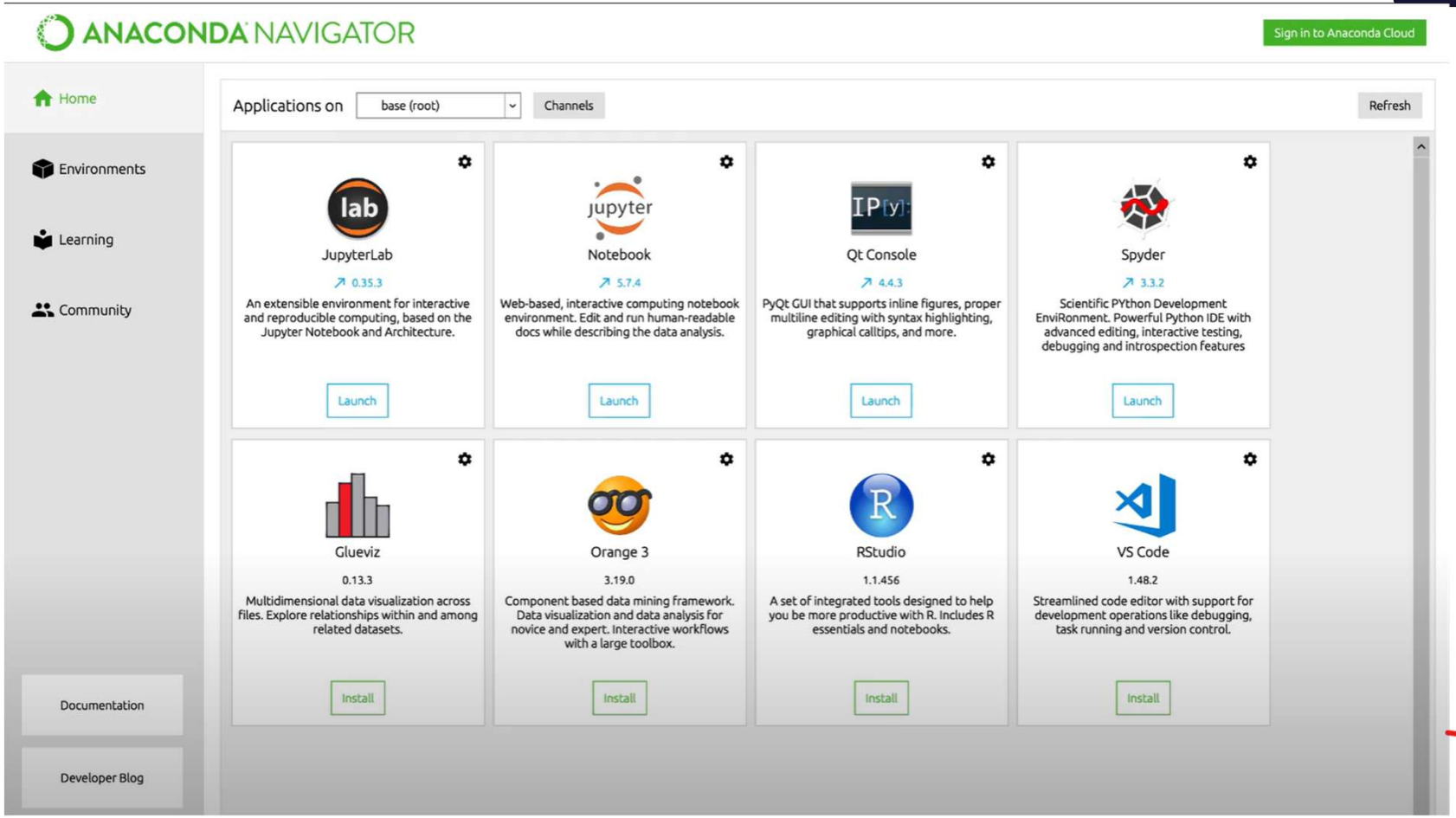
Download and Install Anaconda

- a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment.
- The distribution includes data-science packages suitable for Windows, Linux, and macOS.

  https://www.anaconda.com/download



Anaconda Environment



Execution Mode

Two Basic Python Execution Modes:

- Interactive mode (front-end) a terminal or a console where as soon as you type a python command and press enter it's going to execute immediately. Use JupyterLab or Jupyter Notebook.
- Batch mode (a script) a file editor or an IDLE where you're going to be writing a lot of code and then running all of that code at once. Use like spider or vs code to write some python code and then use the python interpreter to execute all of the code in that file.

Batch Mode

codeworkshop_2023.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  print("Hello, World!")
8
9
10 # In[3]:
11
12
13 print("the world is beautiful~ " * 2)
14
15
16 # In[4]:
17
18
19 name= "Marry"
20 age=6
21
22 print("{} is {} years old".format(name, age))
23
```

Editor

Name	Type	Size	Value
a	int	1	10
age	int	1	6
b	int	1	3
boolean_value	bool	1	False
c	int	1	1000
combined_list	list	7	['pear', 'orange', 1, 2, 3, 4, 5]
dna_seq	str	7	ANTGCTG
ends_with_TG	bool	1	True
float_value	float	1	0.0
fruit	str	6	orange

Help | Variable Explorer | Plots | Files

Variable Explorer

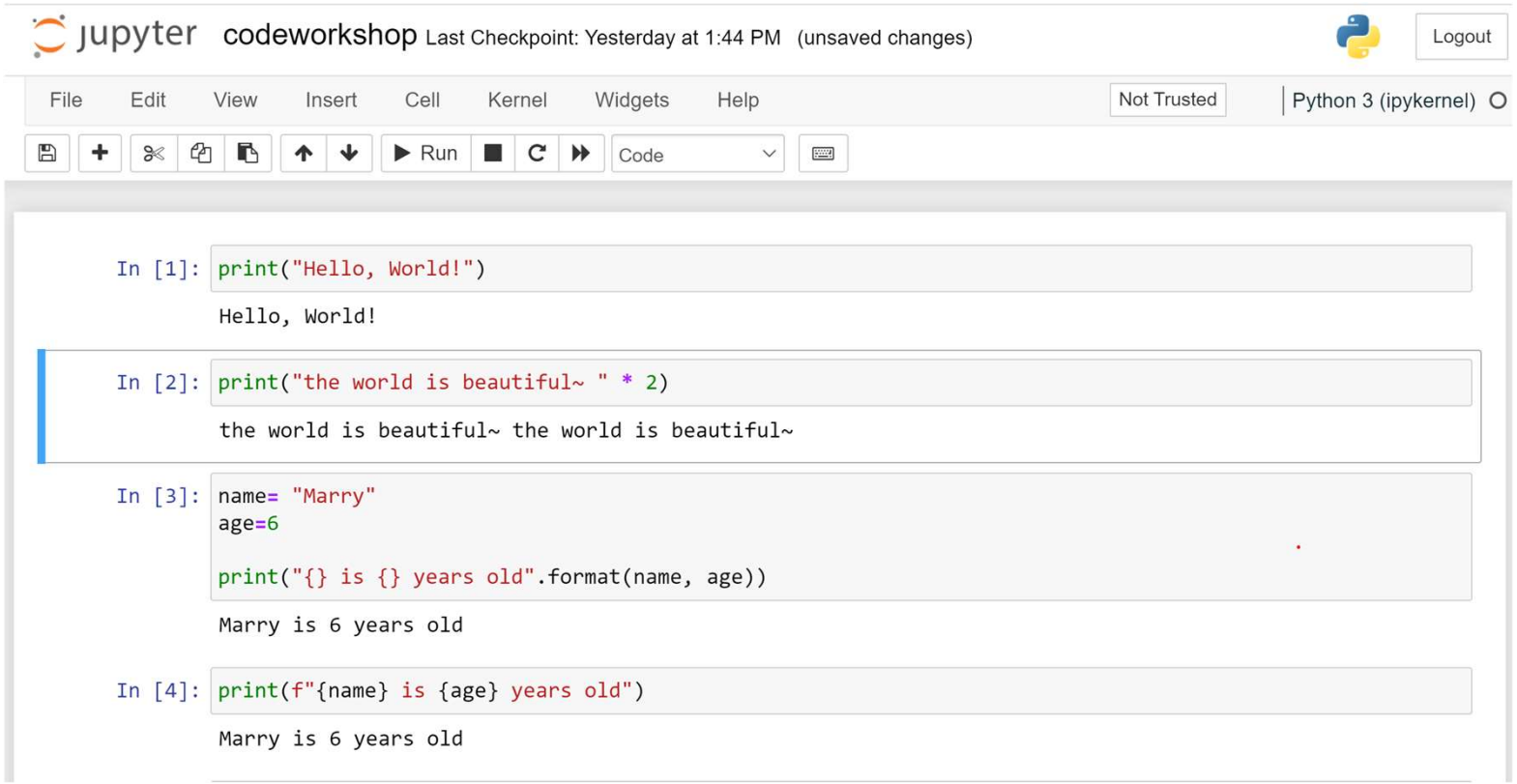
Console 1/A

```
Hello, World!
the world is beautiful~ the world is beautiful~
Marry is 6 years old
Marry is 6 years old
lisa john harris
['lisa', 'john', 'harris']
1
2
3
4
Hello-world
Result of experiment is 20!True
```

Console

https://docs.spyder-ide.org/5/videos/first-steps-with-spyder.html#:~:text=You%20can%20see%20that%20it,open%2C%20edit%20and%20run%20files.

Interactive Mode



https://docs.anaconda.com/ae-notebooks/user-guide/basic-tasks/apps/jupyter/index.html

IDLE (Integrated Development and Learning Environment)

- IDLE is a program used to write and execute Python code
- For windows:
  - Navigate to the Start Menu at the bottom left corner of the screen
  - Type ‘IDLE’ in the search bar and click on the application
- For Mac:
  - Open the Finder and type ‘IDLE’ in the search menu
  - You can also open the Applications folder in the Finder and find the IDLE program

Execution in Interactive Mode

Two Basic Python Modes:

- Interactive mode (front-end)
  - Interactive mode is a command line shell which gives immediate feedback for each statement
  - Executing code in the interactive window happens in 3 steps:
    - 1. Python **reads** the code entered at the prompt (>>>)
    - 2. Python **evaluates** the code
    - 3. Python **prints** the result and waits for more input (>>>)

```
>>>print("Hello, world!")
Hello, world!
>>> 1 + 1
2
```

Python Programming/Batch Mode

Two Basic Python Modes

- Batch Mode (script)
  - The Python Shell (interactive mode) is convenient for executing a few simple commands. If you need to give the computer more complex instructions, it is a good idea to write a script. **A script is a set of instructions you can easily edit and run.**
  - When a program is executed from such a text file, rather than line-by-line in an interactive interpreter, it is called batch mode.
  - 1. Go to **File >>> New File** to create a new script
  - 2. In your script, write: **print("Hello!")**
  - 3. Go to **File >>> Save As…** to save your script. Choose a location and a name. Be sure it saves as a .py file extension
  - 4. Go to **Run >>> Run Module** to run your script. The Python shell will reopen and display: "Hello!"

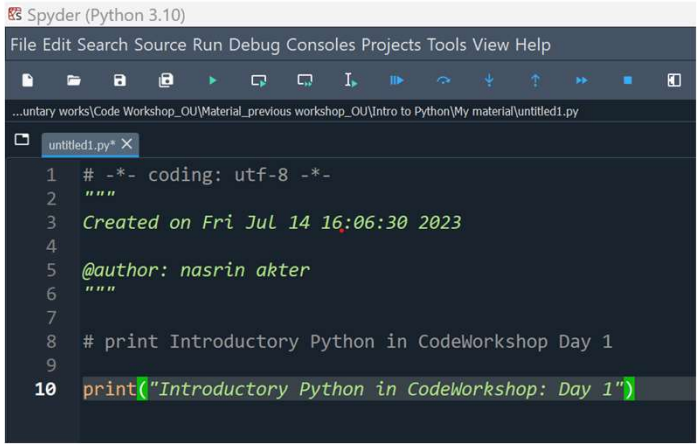
Comments and Docstrings

Programming Best Practices: Commenting and Clarity

- The top priority is making your program easy to understand
  - Makes it possible to verify correct outputs
  - Simplifies modifying and updating
  - Promotes reuse and sharing
- Requirements
  - Docstrings: Block quotes with triple quotation marks are called docstrings. They are used in the interpreter to provide help.
  - Comments: Introduced by using the # symbol. Use comments to describe the steps in your program, helpful notes or clarification, and titles.

Comments and Docstrings

- Comments



- Docstrings

```
n=2
def square(n):
    '''Takes in a number n, returns the square of n'''
    return n**2
print(square(n))
```

```
Console 1/A
In [8]: n=2
   ...: def square(n):
   ...:     '''Takes in a number n, returns the square of n'''
   ...:     return n**2
   ...: print(square(n))
4
```

Python print( ) function

- A Python **function** is a code that performs a task and can be invoked by a name. For example: print( ); sum( ); len( )
- The print( ) function prints the given object to the standard output device (screen)
- You need the print( ) function to see variable values, results, list contents, etc. *** Especially important in batch mode*

Python print( ) function

The function accepts several arguments as shown below with their default values:

```
print(*objects, sep=' ', end= '\n')
```

***objects :** The thing to be printed is represented by the first argument *objects and can be basically anything. It is commonly a string but may be an integer, a float, a variable, a list, or a dictionary, among others. The print() function can also take more than one object.

```
In [1]: print("Hello, World!")
        Hello, World!
```

Normal string operations may be used within the function. As an example, you can multiply a string by an integer as shown here

```
In [2]: print("the world is beautiful~ " * 2)
        the world is beautiful~ the world is beautiful~
```

Python print( ) function

str.format(), where the replacement fields are indicated with curly braces

```
In [4]: name= "Marry"
        age=6

        print("{} is {} years old".format(name, age))
```

```
Marry is 6 years old
```

Similar to this is f-string formatting, but **more compact**

```
In [5]: print(f"{name} is {age} years old")
```

```
Marry is 6 years old
```

starred expression is useful to **print each element** of the list and separated by spaces

```
In [8]: print(*["lisa", "john", "harris"])
```

```
lisa john harris
```

```
In [9]: print(["lisa", "john", "harris"])
```

prints the **entire list** as a single object

```
['lisa', 'john', 'harris']
```

Python print( ) function

Sep: The second argument of the Python print() function defines the separator. If more than one object is given as the first argument, then sep defines what delimiter to place between them.

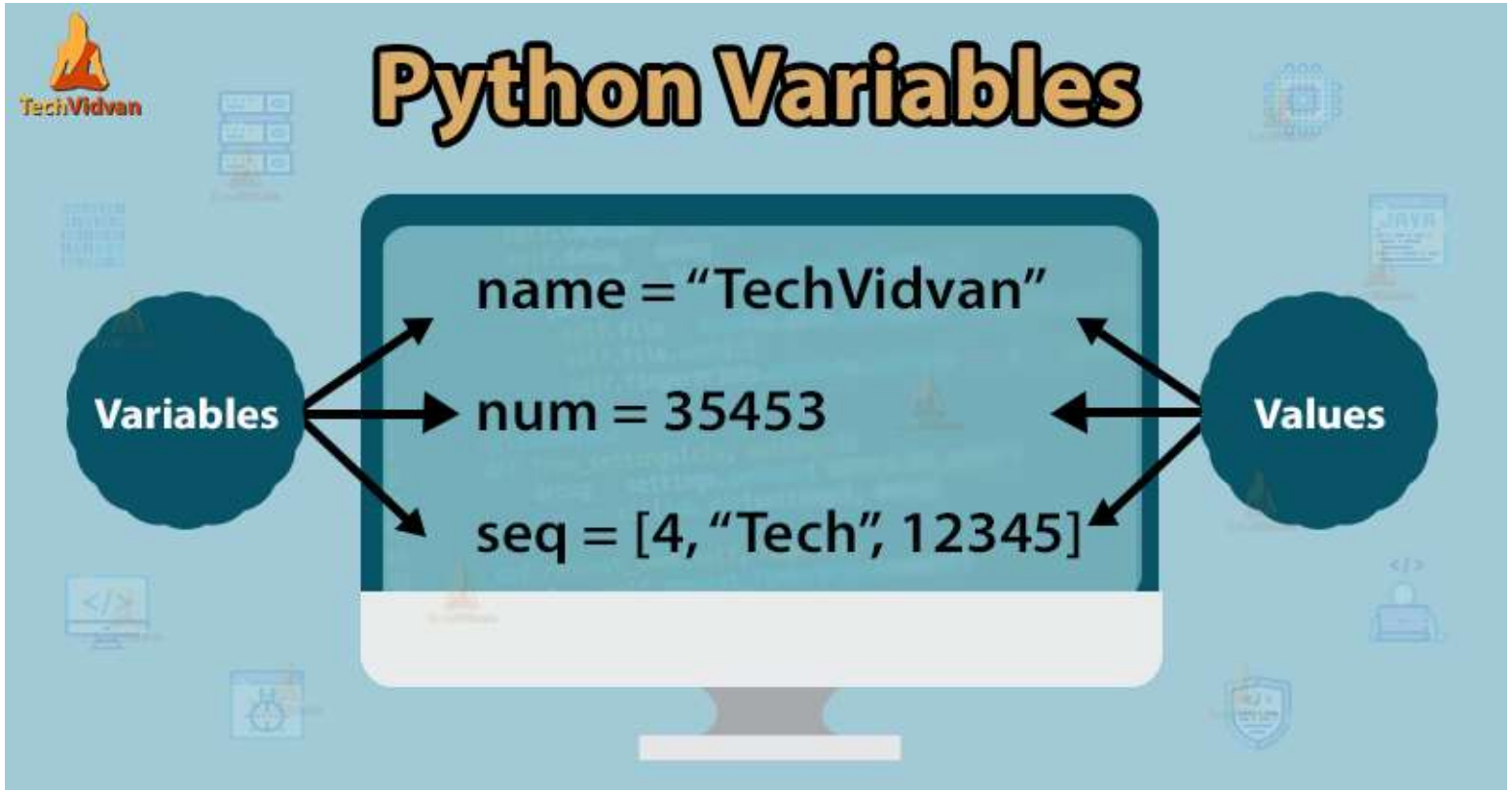
```
In [18]: # \n starts a new line
         print(*[1, 2, 3, 4], sep="\n")

         1
         2
         3
         4
```

```
In [19]: # use a seperator as -
         print("Hello","world", sep = "-")

         Hello-world
```

end: The end keyword allows you to define something to be printed automatically at the end of the output.

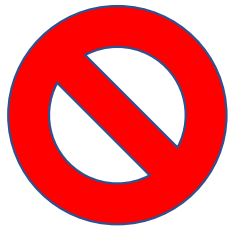
```
In [20]: result=20
         print("Result of experiment", result, sep=" is ", end="!")

         Result of experiment is 20!
```

Variables


TechVidvan
Python Variables
name = "TechVidvan"
num = 35453
seq = [4, "Tech", 12345]
Variables
Values

Variables

- A variable is a value you can change and access throughout your script *(variables are just names)*
- Variable names can only contain these characters:
  - Lowercase letters
  - Uppercase letters
  - Digits (0 through 9)
  - Underscore(_)
- Names cannot begin with a digit
  - 1
  - 1a
  - 1_



Variables: Assign a Variable

- The **assignment operator** is the equal sign (=)
- Pick variable names that are distinct and memorable
  - course = “Python 101”
  - num_students = 30
  - greeting = “Hello world”
- Variables *are case sensitive* (greeting ≠ Greeting)
- Mixed case:
  - numStudents = 30
  - introPythonCourse = “This course is a formal introduction to Python”
  - listOfNames = [“Angie”, “Sanjana”]
- Lower case with underscores*:
  - name_of_class = “Introduction to Python”

Variable Names

Which of the following is a good variable name for a variable holding a value for “meters per second”?

- m/s = 35
- meters_per_second = 35
- metersPerSecond = 35
- m = 35
- speed = 35

Variables: reserved words

- Python has a set of reserved words that cannot be used as variable names

Python Keywords

False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

Basic Variable Types

Numbers

- Integer numbers
  - A whole number (not a decimal)
  - Examples:
    - 100
    - -7
    - 1
    - 1863210897465
- Floating point numbers
  - Numbers that contain a decimal
  - Examples:
    - 3.1416
    - 0.7851
    - 1.0

```
>>> type(1.3)
<class 'float'>
>>> type(5)
<class 'int'>
```

Basic Variable Types

Numbers

- You can check the variable type using the function **type( )**
- You can change the variable type (with certain exceptions) by using the variable type as a function
  - Convert to integer: **int( )**
  - Convert to float: **float( )**

```
a = 5.2
print(a)
print(type(a))

b = int(a)
print(b)
print(type(b))
```



```
5.2
<class 'float'>
5
<class 'int'>
>>>
```

Basic Variable Types

Boolean

- True
- False
- Some functions only return a True or False
  - Example: isinstance*(object, type)*

```
print(isinstance(3.14, float))
print(isinstance(3.14, int))
```

Basic Variable Types

Strings

- A sequence of zero or more characters that are enclosed within a pair of quotation marks.
  - Single quotes, double quotes, triple single quotes, triple double quotes
- Strings are immutable (‘read-only’)
- A string can contain letters, digits, punctuation, white spaces, and other characters.
  - my_string = “Hello world! 123 True”

Basic Variable Types

Typecasting between variable types

From	To	Function	Possible?
Integer	Float	float( )	Yes
Integer	String	str( )	Yes
Integer	Boolean	bool( )	Yes, but…
Float	Integer	int( )	Yes
Float	String	str( )	Yes
Float	Boolean	bool( )	Yes, but…
String	Integer	int( )	Yes, if…
String	Float	float( )	Yes, if…
String	Boolean	bool( )	Yes, but…
Boolean	Integer	int( )	Yes - binary
Boolean	Float	float( )	Yes – binary
Boolean	String	str( )	Yes

Basic Variable Types

Integer to Boolean: Any non-zero integer results in True, while 0 results in False.

```
In [21]: integer_value = 5
         boolean_value = bool(integer_value)
         print(boolean_value)  # Output: True

         integer_value = 0
         boolean_value = bool(integer_value)
         print(boolean_value)  # Output: False
```

```
True
False
```

float to Boolean: Similar approach

```
In [22]: float_value = 3.14
         boolean_value = bool(float_value)
         print(boolean_value)  # Output: True

         float_value = 0.0
         boolean_value = bool(float_value)
         print(boolean_value)  # Output: False
```

```
True
False
```

Basic Variable Types

String to integer: Possible when there is numerical expression.

In [23]:
```
string_value = "123"
integer_value = int(string_value)
print(integer_value)  # Output: 123
print(type(integer_value))  # Output: <class 'int'>
```

```
123
<class 'int'>
```

Operators

- An operator is a symbol that indicates a calculation using one or more operands

Operator	Description	Example
+ Addition	Adds values on either side of the operator	**>>> 5 + 2** **7**
- Subtraction	Subtracts right hand operand from left hand operand	**>>> 5 – 2** **3**
* Multiplication	Multiplies values on either side of the operator	**>>> 5 * 2** **10**
/ Division	Divides left hand operand by right hand operand	**>>> 5 / 2** **2.5**
// Integer Division	The whole number smaller than the floating point result	**>>> 5 // 2** **2**
% Modulus	Integer remainder after b/a	**>>> 5 % 2** **1**
** Exponent	Performs exponential (power) calculation on operators	**>>> 5 ** 2** **25**

Operators

Operator	Description	Example
=	Assigns values from right side expression to left side operand	**c = a + b assigns c the values of a + b**
+= Add AND	It adds right operand to the left operand and assign the result to left operand	**c += a is equivalent to c = c + a** **c += 1 is equivalent to c = c + 1**
-= Subtract AND	It subtracts right operand to the left operand and assign the result to left operand	**c -= a is equivalent to c = c - a**
*= Subtract AND	It multiplies right operand to the left operand and assign the result to left operand	**c *= a is equivalent to c = c * a**
/= Divide AND	It divides left operand to the right operand and assign the result to left operand	**c /= a is equivalent to c = c / a**
//= Integer Division AND	It performs floor division on operators and assign value to the left operand	**c //= a is equivalent to c = c // a**
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	**c %= a is equivalent to c = c % a**
** Exponent	Performs exponential (power) calculation on operators and assign value to the left operand	**c **= a is equivalent to c = c ** a**

Operators

Box Model

Beginner programmers are often puzzled by syntax such as:

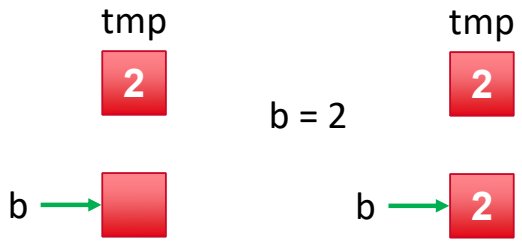
```
b = 2
b = 4 + b
print(b)
```

Before you run this, what do you think the answer is?

Operators

Box Model

- Think of variables as boxes
  - A box is a location in the computer's memory
  - The variable name is a label on the box, or the box's name
  - Assigning b = 2 copies a value from a temporary location into the box



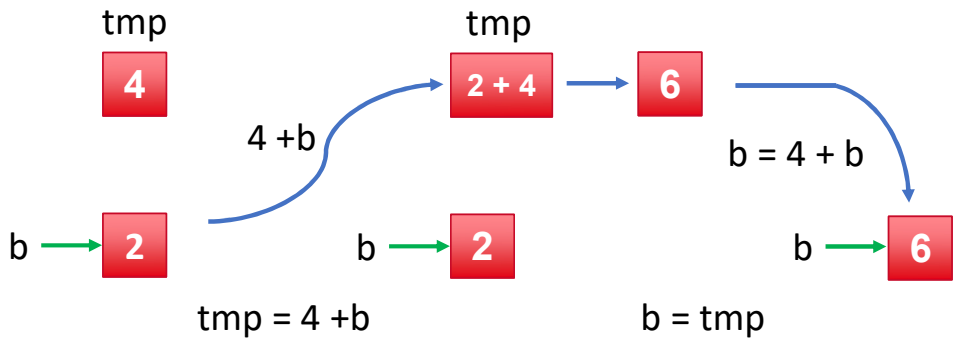
Operators

Box Model

b = 2

b = 4 + b

- 4 is loaded in a temporary space
- b is added to 2 in the temporary space
- The result is copied back into b



Operators

Box Model

- The box model is conceptual
- You can check where the variable is stored using the id( ) function
- Try this:

```
>>>n = 6                    >>>n = n + 4
>>>id(n)                    >>>id(n)
1349663056                 1349663120
```

- The **memory address changes when the value of n is changed**

Operators

Arithmetic Operators:

```
# Arithmetic operators:
a = 10
b = 3

# Addition
c = a + b
print(c)  # Output: 13

# Subtraction
c = a - b
print(c)  # Output: 7

# Multiplication
c = a * b
print(c)  # Output: 30

# Division
c = a / b
print(c)  # Output: 3.3333333333333335

# Floor Division
c = a // b
print(c)  # Output: 3

# Modulo
c = a % b
print(c)  # Output: 1

# Exponentiation
c = a ** b
print(c)  # Output: 1000
```

Operators

Comparison Operators:

```
a = 5
b = 3

# Equal to
result = a == b
print(result)  # Output: False

# Not equal to
result = a != b
print(result)  # Output: True

# Greater than
result = a > b
print(result)  # Output: True

# Less than
result = a < b
print(result)  # Output: False

# Greater than or equal to
result = a >= b
print(result)  # Output: True

# Less than or equal to
result = a <= b
print(result)  # Output: False
```

Operators

Assignment Operators:

```
a = 5

# Addition assignment
a += 3  # Equivalent to: a = a + 3
print(a)  # Output: 8

# Subtraction assignment
a -= 2  # Equivalent to: a = a - 2
print(a)  # Output: 6

# Multiplication assignment
a *= 4  # Equivalent to: a = a * 4
print(a)  # Output: 24

# Division assignment
a /= 6  # Equivalent to: a = a / 6
print(a)  # Output: 4.0

# Modulo assignment
a %= 3  # Equivalent to: a = a % 3
print(a)  # Output: 1.0

# Exponentiation assignment
a **= 2  # Equivalent to: a = a ** 2
print(a)  # Output: 1.0
```

Operators

Logical Operators:

```
a = True
b = False

# Logical AND
result = a and b
print(result)  # Output: False

# Logical OR
result = a or b
print(result)  # Output: True

# Logical NOT
result = not a
print(result)  # Output: False
```

Strings (again)

- Strings are a collection of characters enclosed by quotation marks
- Strings are immutable, meaning they cannot be changed after they are created
- You can use **string methods** to return a new value for the string (but it does not alter the original string)
- It is important to know how characters are indexed in Python
  - alphabet_string = “ABCDEFG”
  - Python indexing starts at 0

String Character	A	B	C	D	E	F	G
Position	0	1	2	3	4	5	6

String Indexing

- You can extract a character using [ ]
- The index will return the character at a given index point
- Look at the following examples: AN0

  dna_seq = “ANTGCTG”

  dna_seq[0]

  dna_seq[5]
- Before running this code, what do you think the results will be?

AN0

```
dna_seq = "ANTGCTG"
print(dna_seq[0])# A
print(dna_seq[5])#T
```

Akter, Nasrin, 2023-07-14T00:38:22.137

String Slicing

- Slicing extracts a series of characters from a string
- The character positions of a slice are specified by two or three integers inside square brackets, separated by a colon

```
dna_seq = "ANTGCTG"
dna_seq[1:4]
```

AN0

Indicates the position of the first character to be extracted

Indicates where the slice ends. **This character will not be included in the slice**

AN0 print(dna_seq[1:4]) # NTG

Akter, Nasrin, 2023-07-14T00:39:36.127

String Concatenation

- String concatenation puts two or more strings together

```
a = "Hello"
b = "world"

print(a + b)
print(a + " " + b)
```

```
================
Helloworld
Hello world
>>>
```

- Can be combined with the input( ) function to create personalized strings.

```
In [33]: name = input("Please enter your name: ")
         print("Hello, " + name + "! Nice to meet you.")

Please enter your name: Nasrin
Hello, Nasrin! Nice to meet you.
```

String Functions

- A function is a piece of code written to carry out a specified task and is called by a name.
- The following are **built-in functions:** print( ); input( ); len( )
- len( ) returns the length of a string:

```
dna_seq = “ANTGCTG”
len(dna_seq)
```

AN0

* Note: using the len( ) function in batch mode will not return a value without adding a print( ) statement

AN0 len(dna_seq) # output7

Akter, Nasrin, 2023-07-14T04:04:15.389

String Methods

1) len(): Returns the length of the string.

```
dna_seq = "ANTGCTG"
length = len(dna_seq)
print(length)  # Output: 7
```

2) lower(): Converts the string to lowercase.

```
lower_seq = dna_seq.lower()
print(lower_seq)  # Output: antgctg
```

3) upper(): Converts the string to uppercase.

```
upper_seq = dna_seq.upper()
print(upper_seq)  # Output: ANTGCTG
```

4) strip(): Removes leading and trailing whitespace from the string.

```
stripped_seq = dna_seq.strip()
print(stripped_seq)  # Output: ANTGCTG
```

5) split(): Splits the string into a list of substrings based on a delimiter.

```
split_list = dna_seq.split("T")
print(split_list)  # Output: ['AN', 'GC','G']
```

6) replace(): Replaces occurrences of a specified substring with another substring.

```
new_seq = dna_seq.replace("TG", "AA")
print(new_seq)  # Output: ANAACAA
```

7) startswith(): Checks if the string starts with a specified substring.

```
starts_with_AN = dna_seq.startswith("AN")
print(starts_with_AN)  # Output: True
```

8) endswith(): Checks if the string ends with a specified substring.

```
ends_with_TG = dna_seq.endswith("TG")
print(ends_with_TG)  # Output: True
```

In-Class Exercise

- dna_seq = “ATgTCtCATTcAAAGCANNNNNATGCGAGTTATGA”

AN0

- Write a simple Python script to:
  - Replace “N” into “G” in the variable dna_seq, and print the sequence
  - Capitalize all the lowercase letters in the variable dna_seq and print the sequence
  - Get the length of dnaseq and print the length
  - Calculate the number of “G” in dna_seq and print the number
- Hints:
  - Write comments as necessary
  - Use string methods and functions

AN0

```
dnaseq = "ATgTCtCATTcAAAGCANNNNNATGCGAGTTATGA"

# Replace "N" with "G"
replaced_seq = dnaseq.replace("N", "G")
print("Replaced Sequence:", replaced_seq)

# Capitalize lowercase letters
capitalized_seq = dnaseq.upper()
print("Capitalized Sequence:", capitalized_seq)

# Get the length of dnaseq
length = len(dnaseq)
print("Length of dnaseq:", length)

# Calculate the number of "G" in dnaseq
count_G = dnaseq.count("G")
print("Number of 'G' in dnaseq:", count_G)
```

Akter, Nasrin, 2023-07-14T04:16:15.463

AN0 0

```
Replaced Sequence: ATgTCtCATTcAAAGCAGGGGATGCGAGTTATGA
Capitalized Sequence: ATGTCTCATTCAAAGCANNNNNATGCGAGTTATGA
Length of dnaseq: 33
Number of 'G' in dnaseq: 5
```

Akter, Nasrin, 2023-07-14T04:32:39.567

Lists

What is a list?

- A list is a collection of characters made from zero or more items, separated by commas, and surrounded by square brackets.
- Examples:
  - empty_list = [ ]
  - weekdays = [“Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”]
  - birds = [‘emu’, ‘ostrich’, ‘cassowary’]
- Items can be of any data type
  - my_list = [123, ‘John’, ‘Terry’, 1.45, True]

Lists


Data Flair
Creating lists
Accessing lists
Slicing lists
Reassigning lists(mutable)
Deleting elements
Multidimensional Lists
Python
Lists
Concatenation of Lists
Operations on Lists
Iterating on a list
List Comprehension
Built-in Functions
Built-in Methods

Lists

# **Creating lists**
fruits = ['apple', 'banana', 'orange']
numbers = [1, 2, 3, 4, 5]
mixed_list = [1, 'apple', True, 3.14]

# **Accessing lists**
print(fruits[0])
# Output: apple

# **Slicing lists-** the last item will be not included
print(numbers[1:4])
# Output: [2, 3, 4]

# **Reassigning lists (mutable)**
fruits[0] = 'pear'
print(fruits)
# Output: ['pear', 'banana', 'orange']

Deleting elements
del fruits[1]
print(fruits)
# Output: ['pear', 'orange']

# **Multidimensional Lists** # let's assume a 3x3 matrix
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(matrix[1][2])
# Output: 6

# **Concatenation of Lists**
combined_list = fruits + numbers
print(combined_list)
# Output: ['pear', 'orange', 1, 2, 3, 4, 5]

Operations on Lists
print(len(fruits)) # Output: 2
print(max(numbers)) # Output: 5
print(min(numbers)) # Output: 1
print(sum(numbers)) # Output: 15

Lists


```
# Iterating on a list
for fruit in fruits:
    print(fruit)
# Output pear orange

# List Comprehension (a concise and powerful way to create lists
squares = [x**2 for x in numbers]
print(squares)
# Output: [1, 4, 9, 16, 25]

# Built-in Functions
print(len(fruits))
# Output: 2
print(sorted(numbers))
# Output: [1, 2, 3, 4, 5]
print(sum(numbers))
# Output: 15
```

```
# Built-in Methods
fruits.append('grape')
print(fruits)
# Output: ['pear', 'orange', 'grape']

fruits.remove('pear')
print(fruits)
# Output: ['orange', 'grape']

fruits.insert(1, 'apple')
print(fruits)
# Output: ['orange', 'apple', 'grape']
```

Lists

How to access list elements or items:

- As with strings, you can extract a single value from a list by indexing.

```
>>>sequences = [“AAA”, “TTT”, “GGG”]
>>>sequences[0]
        ‘AAA’
>>>sequences[1]
        ‘TTT’
```

Changing a List / Changing a String

Lists

- Lists are **mutable**

```
>>>sequences = [“AAA”, “TTT”, “GGG”]
>>>sequences[2] = “CCC”
>>>print(sequences)
```

[‘AAA’, ‘TTT’, ‘CCC’]

Strings

- Strings are **immutable**

```
DNAseq = “ANTGCTG”
DNAseq[1] = “G”
```

Returns a **type error**

Dictionaries

- Dictionaries are a powerful data structure in Python that store data in key-value pairs.
- Unlike lists, which use numeric indices to access elements, dictionaries use keys for accessing values.
- Keys can be of any immutable type (e.g., strings, numbers, tuples).
- Values can be of any data type, including other dictionaries.

Dictionaries


```
my_dict = {
    "key1": "value1",
    "key2": "value2",
    "key3": "value3"
}
```

- Use curly brackets to create dictionaries.
- Use colon in between keys and values

Conditional Statements

- Conditional statements in Python allow us to make decisions based on certain conditions.
- We use if and else to create conditional statements.

```
if condition:
    # code block executed if condition is True
else:
    # code block executed if condition is False
```

Loops

- Loops in Python allow us to execute a block of code repeatedly.
- Use loops to iterate over lists, strings, dictionaries, and other iterables.
- Two types of loops: for loop and while loop.

```
for item in iterable:
    # code block executed for each item in the iterable
```

```
while condition:
    # code block executed repeatedly as long as the condition is True
```

Functions

- Functions are blocks of reusable code that perform specific tasks.
- They help in organizing code and making it more modular.
- Functions can have multiple parameters and return multiple values.
- Functions are called using the function name followed by parentheses: function_name().
- Use functions to avoid repetitive code and improve code readability.

Functions


```
def function_name(parameters):
    # code block of the function
    return result
```

def: keyword to define a function.

function_name: the name of the function.

parameters: optional input to the function.

return: statement to return a value from the function.

Any Questions