

COP 3035

Intro Programming in Python

Summer 2024

Lecture 2 – part 1

Lab1 - Due Date: 05/20/2024

Homework 1 - Due date: 05/24/2024

Lecture 2 – part 2

Review

Review

What is Python?

Who created Python, and why?

How is Python ranked among other languages?

What is the official documentation webpage for Python?

What topics will we cover in this course?

What are the different ways to run Python code?

What is the difference between Jupyter notebooks and an IDE?

How can you do basic arithmetic with python ?

Tools Review

Python Shell,

Anaconda:

- Anaconda Cloud
- Anaconda Navigator

Jupyter Notebooks,

Jupyter Lab,

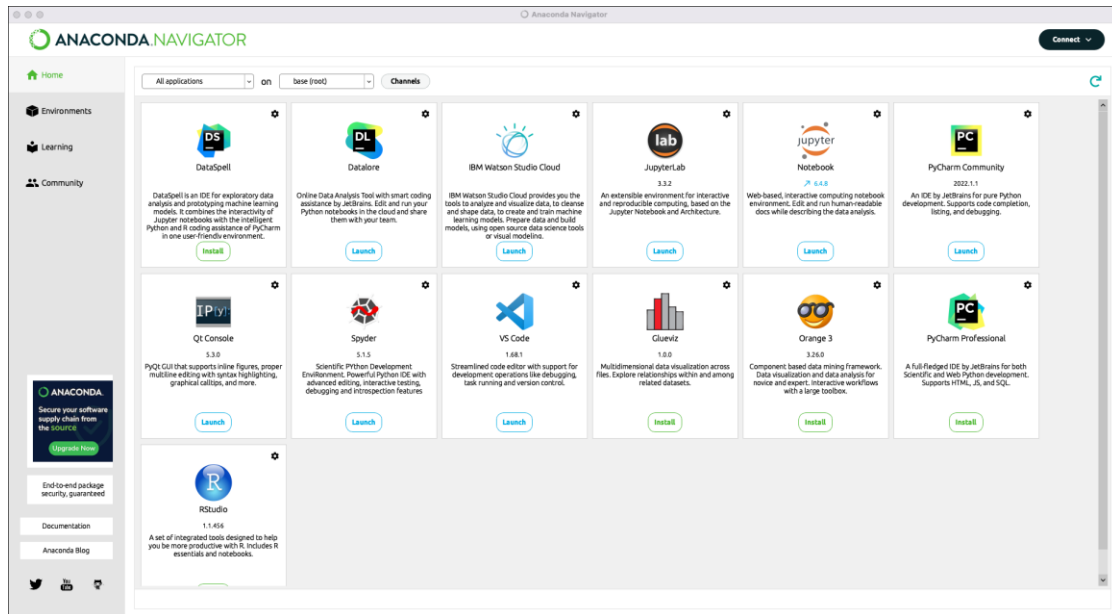
Google Collab,

IDE – Integrated Development Environment,
Visual Studio,

Others

Anaconda Navigator (Local PC)

<https://docs.anaconda.com/free/navigator/index.html>

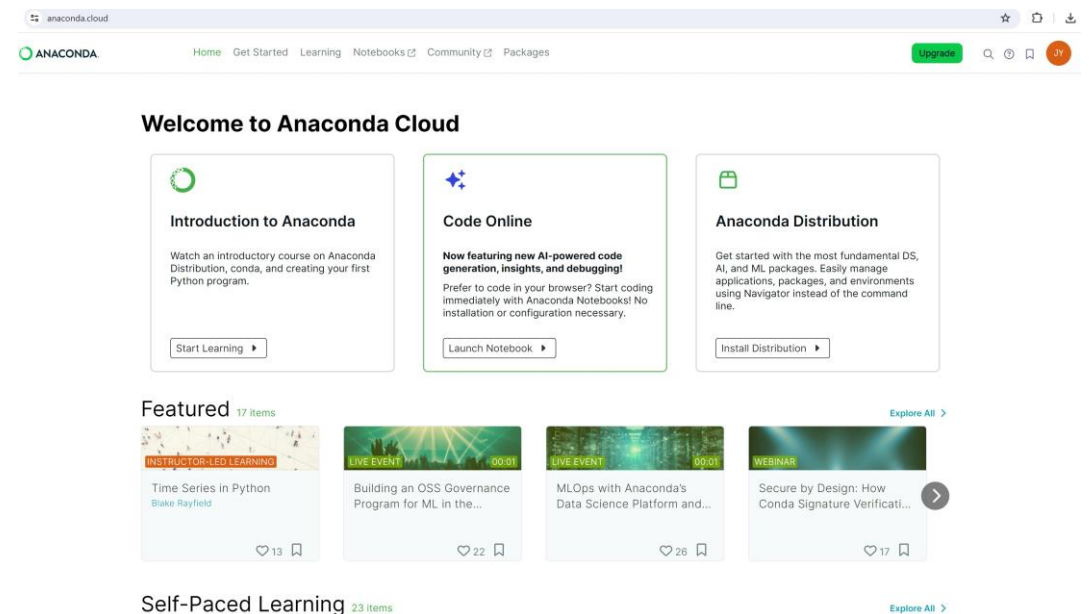


<https://docs.anaconda.com/free/navigator/install/>

<https://docs.anaconda.com/free/navigator/tutorials/create-python35-environment/>

Anaconda Cloud (cloud server)

<https://anaconda.cloud/>



Lecture 2 – part 3

Numbers, Variables, Strings

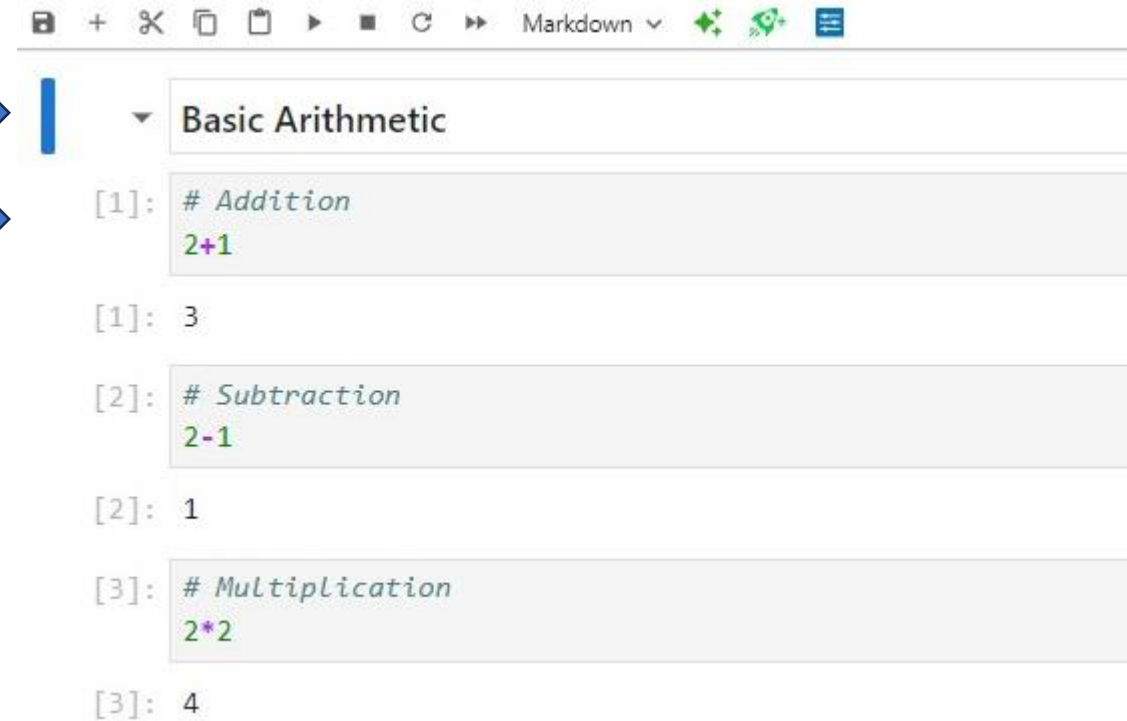
Basic Notebooks

- Basic Arithmetic
- Variable Assignments



Markdown cell →

Code cell →



Dynamic Typing

- **Dynamic typing** in Python refers to the ability of the language to determine the type of a variable at runtime rather than at compile time.
- This means that you do not need to explicitly declare the type of a variable when you create it; instead, the type is inferred from the value assigned to the variable.

Dynamic typing makes Python a very flexible and easy-to-use language, **but** it also requires developers to be careful with their type assumptions and usage to avoid runtime errors.

```
x = 10          # x is an integer
x = "hello"     # now x is a string

y = 5           # y is initially an integer
y = [1, 2, 3]   # now y is a list

z = 3
z = z + "world" # Is this valid ?
```

The **type()** function in Python is used to determine the type of an object at runtime.

Strings


- A string in Python is a sequence of characters.
- Strings are immutable, meaning they cannot be changed after creation.
- To create a string in Python you need to enclose characters in single quotes, double quotes, or triple quotes (multiline).

```
text = "This is a string"
```

String Concatenation

- Combine strings using the + operator

```
greeting = "Hello" + " " + "World"    # Output: "Hello World"
```

Two blue arrows pointing upwards from below the code line to the spaces between the string literals "Hello", " ", and "World".

- We can use the multiplication symbol to create repetition!

```
letter = "z"  
Letter*10  
'zzzzzzzzzzzz'
```

Escape Sequences

- An escape sequence is a sequence of characters
- When used inside a character or string, does not represent itself but is converted into another character or series of characters that may be difficult or impossible to express directly, like newline (`\n`), tab (`\t`), and so on.

List of Escape Sequence Available in Python	
Escape Sequence	Meaning
<code>\'</code>	Single quote
<code>\\'</code>	Double quote
<code>\\</code>	Backslash
<code>\n</code>	Newline
<code>\r</code>	Carriage Return
<code>\t</code>	Horizontal Tab
<code>\b</code>	Backspace
<code>\f</code>	Formfeed
<code>\v</code>	Vertical Tab
<code>\0</code>	Null Character
<code>\N{Name}</code>	Unicode character Database named lookup
<code>\uxxxxxxxx</code>	Unicode character with a 16-bit hex value
<code>\Uxxxxxxxx</code>	Unicode character with a 32-bit hex value
<code>\000</code>	Character with octal value 000
<code>\xhh</code>	Character with hex value hh

<https://www.scaler.com/topics/escape-sequence-in-python/>

String Indexing

- Strings are sequences of characters, and each character has a **position or index**.
- Indexing **starts at 0** for the first character and goes up to **len(string) - 1** for the last character.
- Use square **brackets []** to access a character at a specific index.
- **Negative Indexing:** Use negative indices to access characters from the end of the string.

```
s = "Hello"  
s[0]   # Output: 'H'  
s[1]   # Output: 'e'
```

```
s = "Hello"  
s[-1]  # Output: 'o'  
s[-2]  # Output: 'l'
```

String Slicing

- Slicing allows you to obtain a substring by specifying a range of indices.

Syntax:

`string[start:stop]`

`start` is the index where the slice begins (inclusive).

`stop` is the index where the slice ends (exclusive).

```
s = "Hello World"  
s[0:4]
```

String Methods

Method	Description	Example
str.lower()	Converts all characters in the string to lowercase.	"Hello".lower() → "hello"
str.upper()	Converts all characters in the string to uppercase.	"Hello".upper() → "HELLO"
str.strip()	Removes leading and trailing whitespace.	" Hello ".strip() → "Hello"
str.split()	Splits the string into a list of substrings.	"Hello World".split() → ["Hello", "World"]
str.join(iterable)	Joins elements of an iterable into a single string.	"-".join(["Hello", "World"]) → "Hello-World"
str.replace(old, new)	Replaces occurrences of a substring.	"Hello World".replace("World", "Python") → "Hello Python"
str.find(sub)	Returns the lowest index of the substring.	"Hello".find("e") → 1
str.startswith(prefix)	Checks if the string starts with a prefix.	"Hello".startswith("He") → True
str.endswith(suffix)	Checks if the string ends with a suffix.	"Hello".endswith("lo") → True
len(obj)	Returns the length of an object.	len("Hello") → 5