Session 1_all

July 11, 2025

1 Python as a calculator

[ ]:
```
2+1
```

[ ]:
```
2**3   # exponentiation
```

[ ]:
```
3/10
```

1.1 Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

[ ]:
```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print(counter)
print(miles)
print(name)
```

[ ]:
```
a = 1
b = 2
print(a,b)
```

[ ]:
```
# Arithmetic with variables
print(a + b)
print(a + 10)
print(a/b)
```

[ ]:
```
# create variable c
c = a + b
print(c)
```

1.2 Class Exercise

Why does this code not work?

[ ]:
```
my_variable = 10
print(my_varable)
```

2 Standard Data Types

2.1 Basic data types in Python

We have seen two Python data types:

- `int`, or integer: a number without a fractional part. e.g. 1,2,100...
- `float`, or floating point: a number that has both an integer and fractional part, separated by a point, e.g. 1.2, 1.3, 28.9...
- `str`, or string: a type to represent text. You can use single or double quotes to build a string, e.g. `'Lucy'`, `'Seattle'`.
- `bool`, or boolean: a type to represent logical values. Can only be `True`, or `False`.

2.2 Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]).

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

[ ]:
```
# A list is a compound data type, you can group values together
a = 10
my_list = [ a, 'abcd', 786 , 2.23, 'john']
print(my_list)
```

[ ]:
```
tinylist = [123, 'john']

print(tinylist * 2)       # Prints list two times
print(my_list + tinylist) # Prints concatenated lists
```

2.2.1 Indexing and Slicing

[ ]:
```
print(my_list)          # Prints complete list
print(my_list[0])       # Prints first element of the list
print(my_list[1:3])     # Prints elements starting from 2nd till 3rd
print(my_list[2:])      # Prints elements starting from 3rd element
print(my_list[2:-1])
print(my_list[-1])      # Prints the last element
print(my_list[-3:-1])
```

[ ]:
```
# Variables can also be used as indexing
myStart = 2
print(my_list[myStart])

# get the length of a list
len(my_list)
```

2.2.2 Practice

A digital marketing manager tracked their daily ad spend for a campaign over one week. The list ad_spend contains alternating entries of a day of the week and the dollar amount spent on ads for that day.

`ad_spend = ["Monday", 120.0, "Tuesday", 150.5, "Wednesday", 135.0, "Thursday", 160.0, "Friday", 175.5, "Saturday", 190.0, "Sunday", 145.0]`

- Use slicing to create a list, `weekday_spend`, that contains the ad spend data for Monday to Friday.
- Use slicing to create a list, `weekend_spend`, that contains the ad spend data for Saturday and Sunday.
- Print both `weekday_spend` and `weekend_spend`.

2.2.3 List Manipulation

Replacing list elements is pretty easy. Simply subset the list and assign new values to the subset. You can select single elements or you can change entire list slices at once.

[ ]:
```
x = ['a', 'b', 'c', 'd']
x[1] = 'r'
x[2:] = ['s', 't']
print(x)
```

Extend a list with the + operator.

[ ]:
```
x = ['a', 'b', 'c', 'd']
y = x + ['e', 'f']
print(y)
```

3 Packages

For example, you may want to calculate the area of a circle. When the radius is `r`, the area is $A = \pi r^2$.

To use the constant `pi`, you'll need the `math` package.

[ ]:
```
# radius
r = 0.3
import math      # Import the math package.
                 # Now you can access the constant pi with math.pi
```

```
A = math.pi *r *r
print(A)
```

3.1 Selective import

General imports, like `import` math, make all functionality from the `math` package available to you. However, if you decide to only use a specific part of a package, you can always make your import more selective.

[ ]:
```
from math import pi
A = pi *r *r
print(A)
```

3.2 Practice

Selectively import the square root function `sqrt` from `math` package. Then compute the square root of 10.

4 Numpy

`NumPy` (Numeric Python) is the fundamental package for data science and scientific computing with Python.

[ ]:
```
# the list length represents the length of some rectangles
# the list width represents the width of some rectangles
length = [2, 4, 6, 8]
width = [1.4, 3.2, 1, 6]

# we want to calculate area of the rectangles
length * width
# or double the length of every rectangle
length * 2
```

Numpy array is an alternative to Python list. It makes calculations over the entire arrays.

[ ]:
```
import numpy as np    # you can now refer to numpy as np
# use np.array() to create a numpy array from length.
np_length = np.array(length) # np.array is a function to do the type conversion
np_length
```

[ ]:
```
# Math on numpy arrays
print(np_length+1)
print(np_length*2)
print(np_length + np_length)

np_width = np.array(width)
print(np_length * np_width)
```

4.1 Practice

You're analyzing the performance of a digital marketing campaign across 5 different countries. Your goal is to calculate the **Return on Investment (ROI)** for each country.

- You ran ads in 5 countries. The **ad spend** (in USD) is: `ad_spend = [1200, 950, 1600, 870, 1320]`. Let's first convert this list into a NumPy array called `spend_usd`.
- Each country generated revenue in its local currency: `revenue_local = [3000, 2200, 4100, 1900, 3400]`.
- The excange rate to USD for each country is `exchange_rates = [1.0, 0.85, 1.1, 0.9, 1.2]  # Local → USD`
- Convert `revenue_local` into USD by multiplying it with `exchange_rates` element-wise. Store the result in a new array called `revenue_usd`.
- Calculate ROI for each country:

$$ROI = \frac{Revenue(USD) - Spend(USD)}{Spend(USD)}$$

- Store the result as a NumPy array called `roi` and print it out.

4.2 Numpy array indexing and slicing

To subset both regular Python lists and numpy arrays, you can use square brackets:

[ ]:
```
x = [4 , 9 , 6, 3, 1]
print(x[1])
y = np.array(x)
print(y[1])
print(y[1:3])
print(y[3:])
print(y[-2])
```

For `numpy` only, you can also use logic values (True, False).

[ ]:
```
my_index = np.array([True, True, False, False, True])
y[my_index]
print(y[my_index])
```

This is most often used with comparative operators to select items:

[ ]:
```
high = y > 5
y[high]
```

Logical operators in Python:

Operator	Description
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to

[ ]:
```
y == 1
y != 1
```

For numpy only, you can also use a list of integers to select positions.

[ ]:
```
print(y)                # print the entire numpy array
sub_y = y[[0,4,2]]      # get a subset of the numpy array
                        # and assign it to sub_y
print(sub_y)            # print the subset

# compare with list
print(x)
#t = x[[0,4,2]]         # wrong syntex
```

4.3 Comparing Numpy with basic Python lists

Numpy is great for doing vector arithmetic. If you compare its functionality with regular Python lists, however, some things have changed.

- Numpy arrays cannot contain elements with different types. If you try to build such a list, some of the elements' types are changed to end up with a homogeneous list.
- The typical arithmetic operators, such as + and * have a different meaning for regular Python lists and numpy arrays.
- Element-wise Operations (NumPy only).
- Boolean and subset Indexing (NumPy only).

4.4 2D numpy arrays

Now let's construct our first 2D array. Suppose you're a sales analyst at a company. You have quarterly sales data (in $1,000s) for 2 products (A, B) across 4 regions (North, South, East, West).

```
[ ]: product_A = [250, 300, 280, 320]
     product_B = [180, 200, 190, 210]

     # Rows: Products A, B
     # Columns: Regions North, South, East, West

     list_2d = [[250, 300, 280, 320], [180, 200, 190, 210]]
     np_2d = np.array(list_2d)
     np_2d
```

0	1	2	3	
250	300	280	320	0
180	200	190	210	1

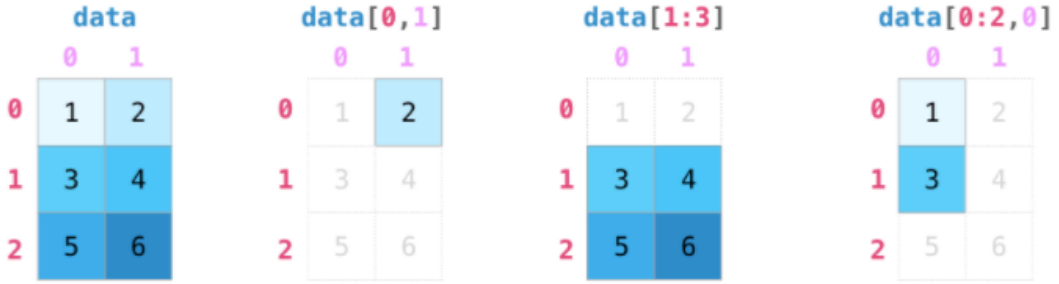
```
[ ]: np_2d.shape    #2 rows and 4 columns
```

4.4.1 Indexing and Slicing

```
[ ]: np_2d[0,2]
```

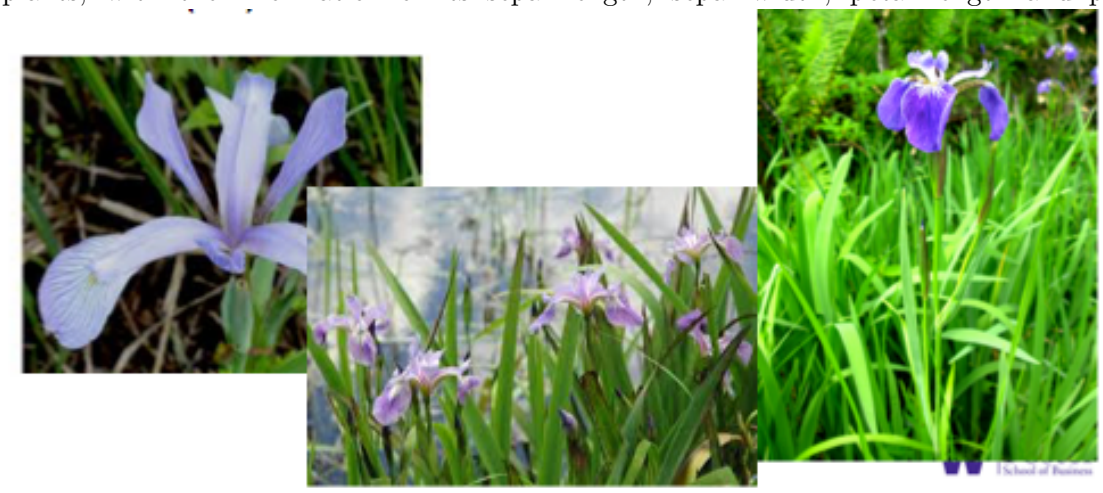
```
[ ]: # Select the height and weight of the third and fourth player
     np_2d[:,1:3]   # : stands for all the rows
     # product A of the second and third region
     np_2d[0,1:3]
```

```
[ ]: # If you only want to retrieve the sales for product A
     np_2d[0,:] # it is equivalent to np_2d[0] which omits the column index
     np_2d[0, [0,2,3] ]
```



4.4.2 Practice

We will use the iris dataset. This is perhaps the best known database to be found in the pattern recognition and data mining literature. This dataset contains 150 different iris plants, with the information of its sepal length, sepal width, petal length and petal width.



[ ]:
```
from sklearn.datasets import load_iris
iris = load_iris()                # load iris dataset
print(iris.DESCR)                  # get the description of the dataset
```

[ ]:
```
iris_attr = iris.data      # numpy array contains attributes sepal length,
                           # sepal width, petal length and petal width
species = iris.target      # numpy array contains the type of each iris
print(iris.feature_names)
```

Questions:

1. Print out the shape of `iris_attr`.
2. Print out the 3rd row of `iris_attr`.
3. Make a new variable, sepal width, containing the entire second column of `iris_attr`.
4. What is the petal length (the 3rd column) of the 95th iris plant?
5. Find observations whose species are "versicolor" (whose target value is 1).
6. Find out the average petal length of all versicolors using `np.mean()`.