



Machine Learning

Dr. Muhammad Adeel Nisar

Assistant Professor – Department of IT,
Faculty of Computing and Information Technology,
University of the Punjab, Lahore

Recap of the Last Lecture

- Applications of Machine Learning, Based on
 - Images and Videos Data
 - Sensor Data
 - Audio Data
 - Tabular Data
 - Text Data
- Machine Learning Life-Cycle

Today's Contents

- Types of Machine Learning
- Supervised Machine Learning
- Regression
 - Hypothesis Function
 - Cost Computation
 - Optimization Mechanism
- Introduction to Python (Lab)

Types of Machine Learning

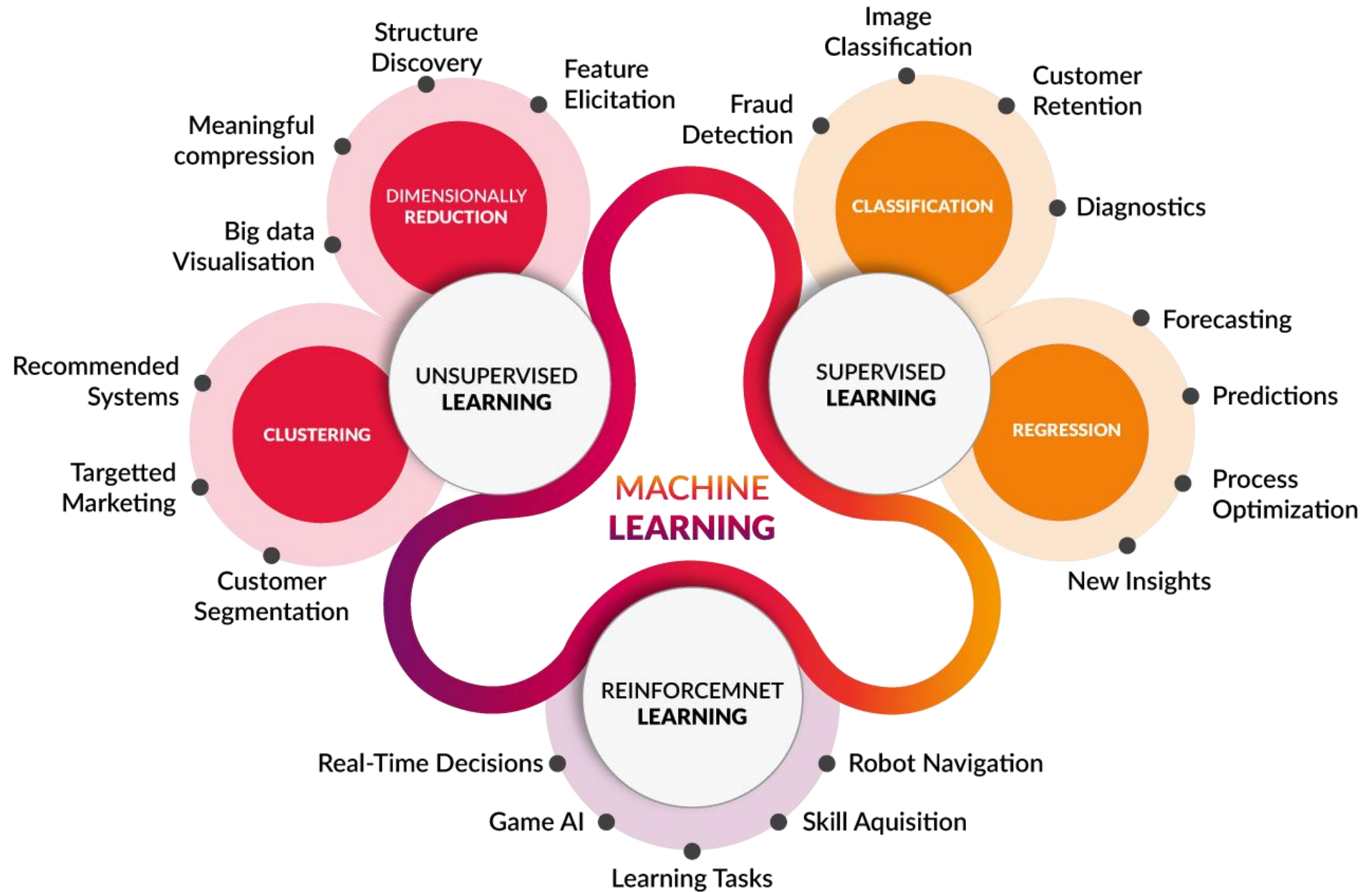
- **Supervised Machine Learning** assumes that a set of labelled training data $\{(x^{(i)}, y^{(i)})\}$ where i is 1 to m is available and the classifier is designed by exploiting this a-priori known information.
- Two further types
 - Regression
 - Linear Regression
 - Nonlinear Regression
 - Classification
 - Logistic Regression
 - Naïve Bayes
 - Support Vector Machines etc

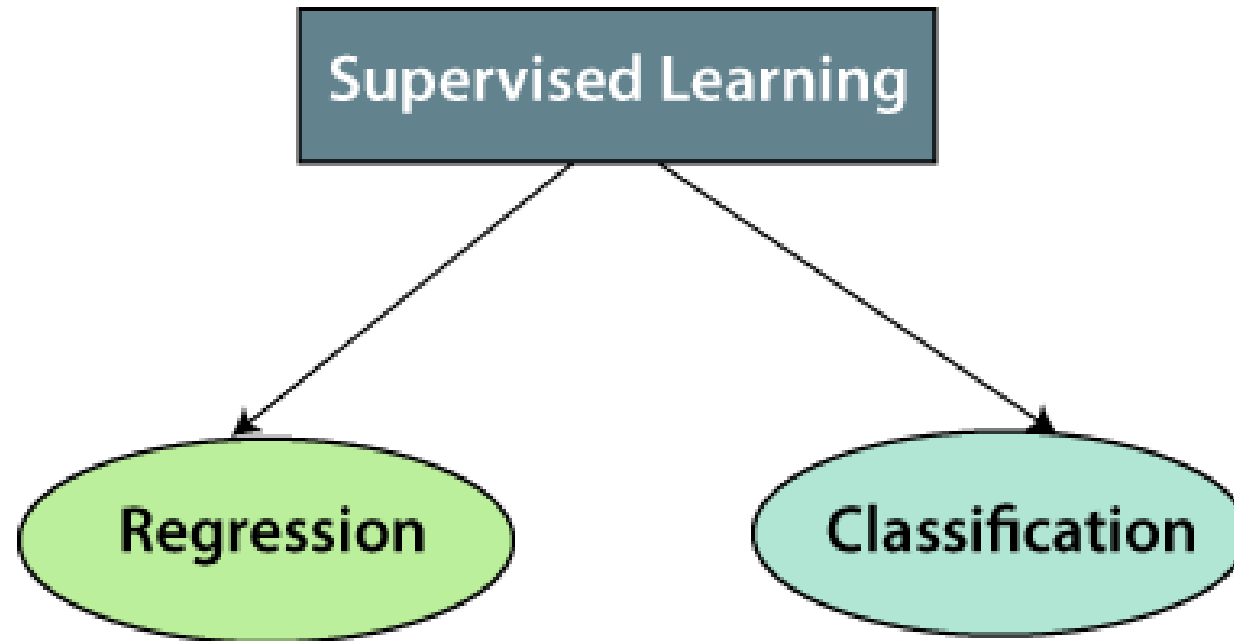
Types of Machine Learning

- **Unsupervised Machine Learning** clusters unlabeled training data, $\{\underline{x^{(i)}}\}$ where i is 1 to m , described by feature vectors, $x^{(i)}$, into similar groups
 - Clustering
 - K-Means Clustering
 - Dimensionality Reduction
 - Principal Component Analysis
 - Autoencoders

Types of Machine Learning

- In **Semi-supervised Machine Learning** the dataset contains both labeled and unlabeled examples. Usually, the quantity of unlabeled examples is much higher than the number of labeled examples. The goal of a semi-supervised learning algorithm is the same as the goal of the supervised learning algorithm.
- **Reinforcement Learning** solves a particular kind of problems where decision making is sequential, and the goal is long-term, such as game playing, robotics, resource management, or logistics.





Regression

- A Supervised learning algorithm
- Taking input variables and trying to fit the output onto a continuous values.
- Linear regression with one variable is also known as “Univariate linear regression”.
- Univariate linear regression is used when you want to predict a single output value y from a single input value x .
- The Hypothesis Function $y' = h\theta(x) = \theta_0 + \theta_1 x$
- Given the training data with right answers, predict the real-valued output for the test data.

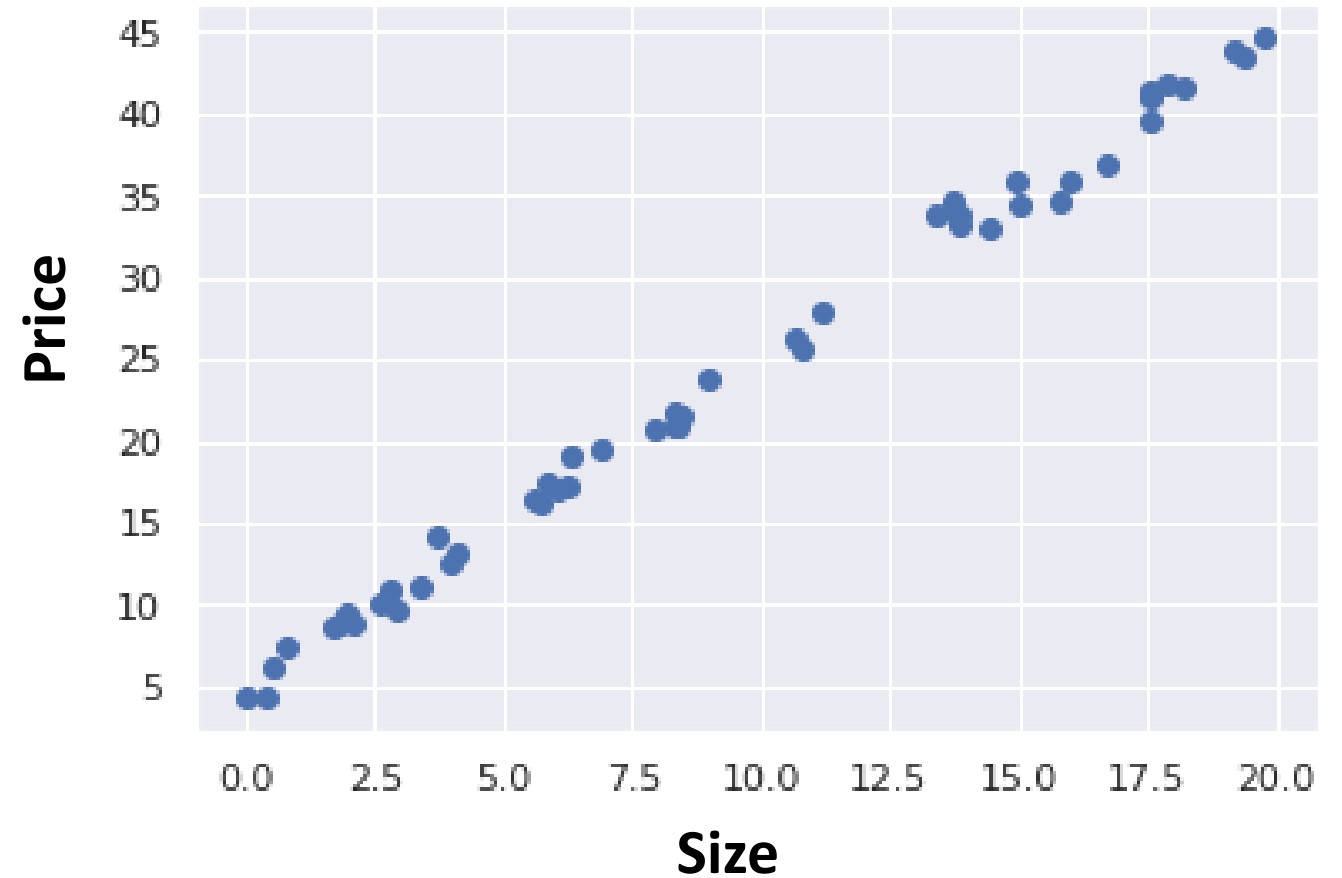
Dataset and Notations

- Notations
- m = Number of Training Examples
- \mathbf{x} = Input Variable/ Features
- \mathbf{y} = Output Variable / Target Value
- (\mathbf{x}, \mathbf{y}) is one training example
- $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ is i^{th} training example
- $(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}) = (8.3, 20.99)$

	Input Data (x)	Correct Answer (y)
1	8.3	20.99
2	14.4	32.89
3	6.05	17.1

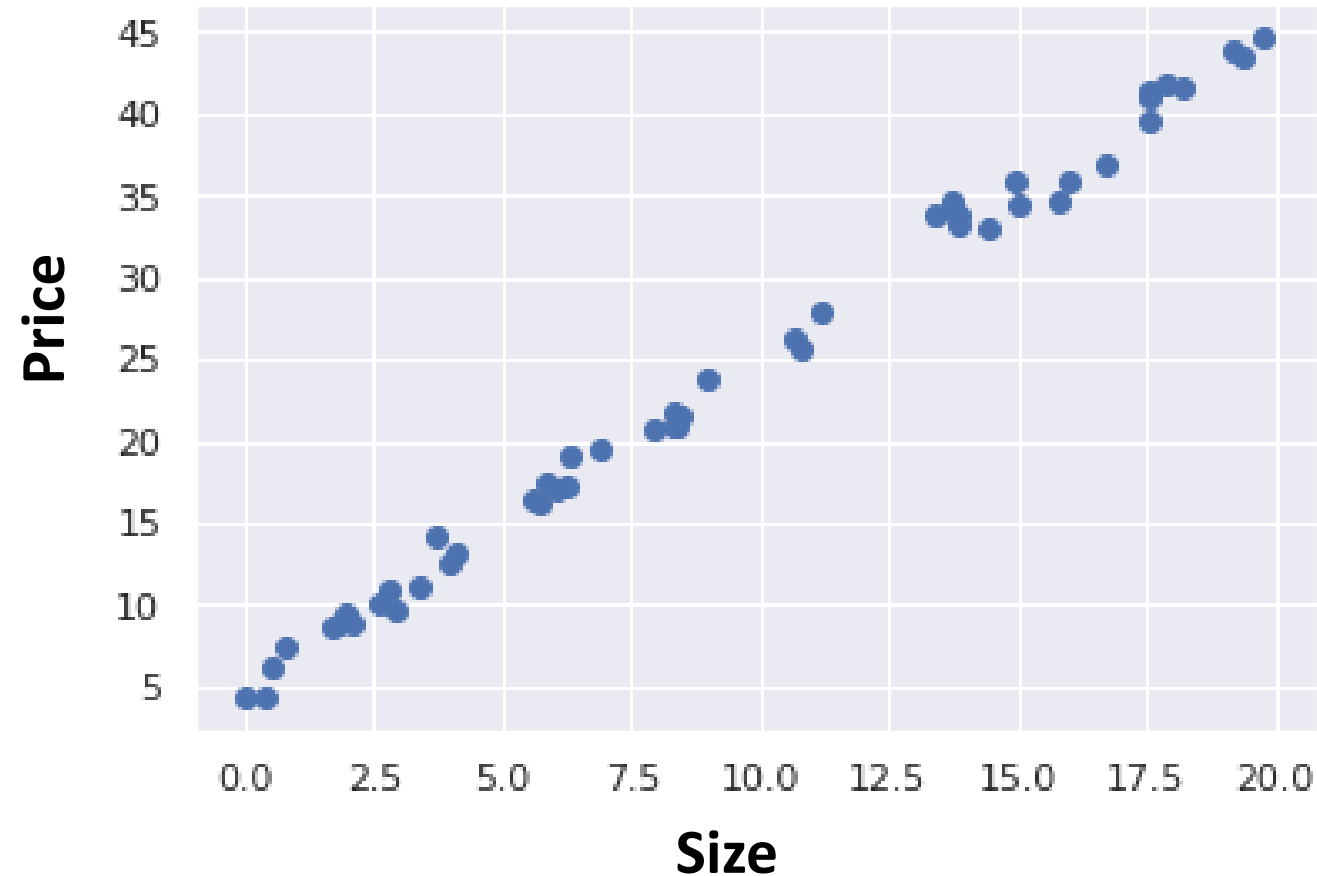
Dataset and Plotted Graph

Input Data (x)	Correct Answer (y)
8.3	20.99
14.4	32.89
6.05	17.08
..	..



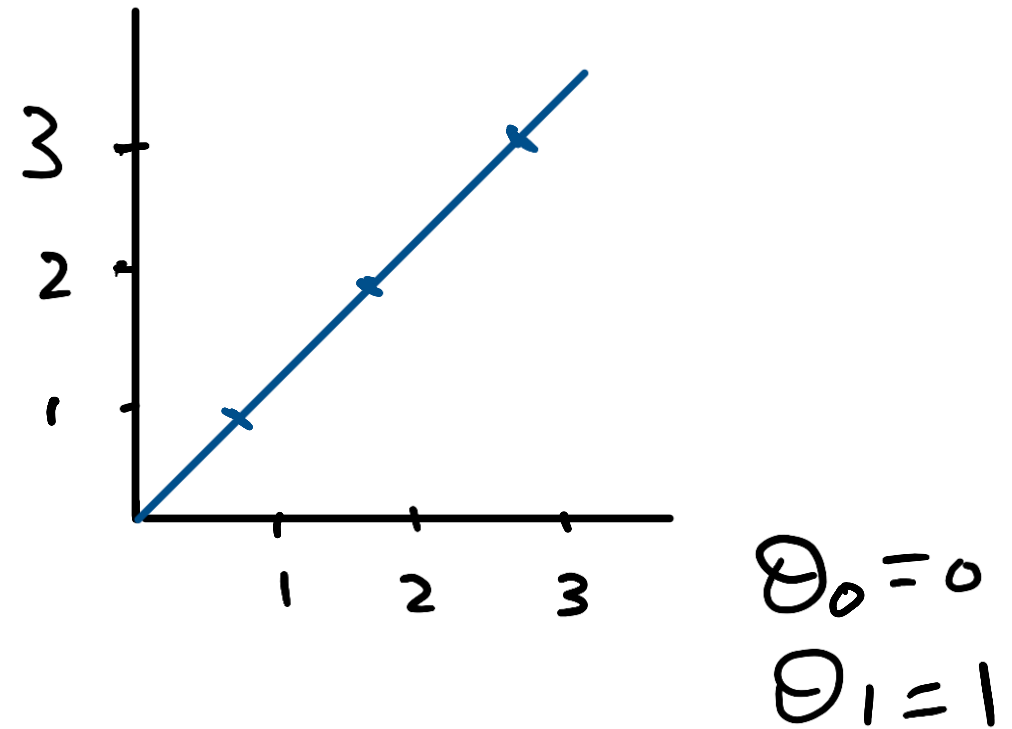
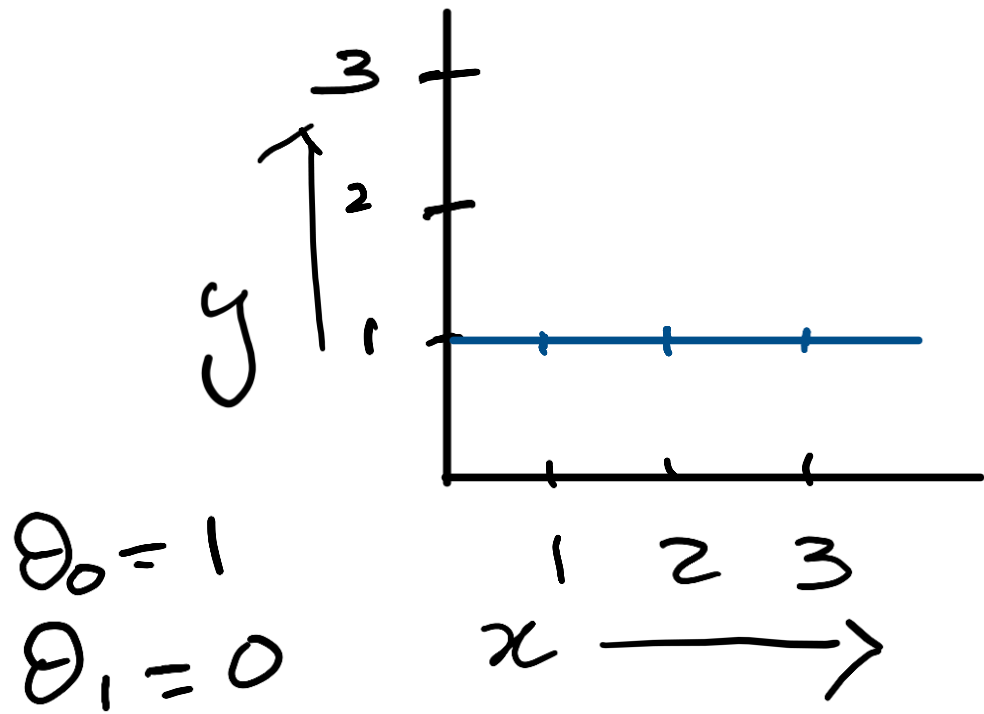
Linear Regression with One Variable

- This is like the equation of a straight line.
- We give $h\theta(x)$ values for θ_0 and θ_1 to get our estimated output y' .
- We are trying to create a function that will map out input data to our output data.



Equation of Line

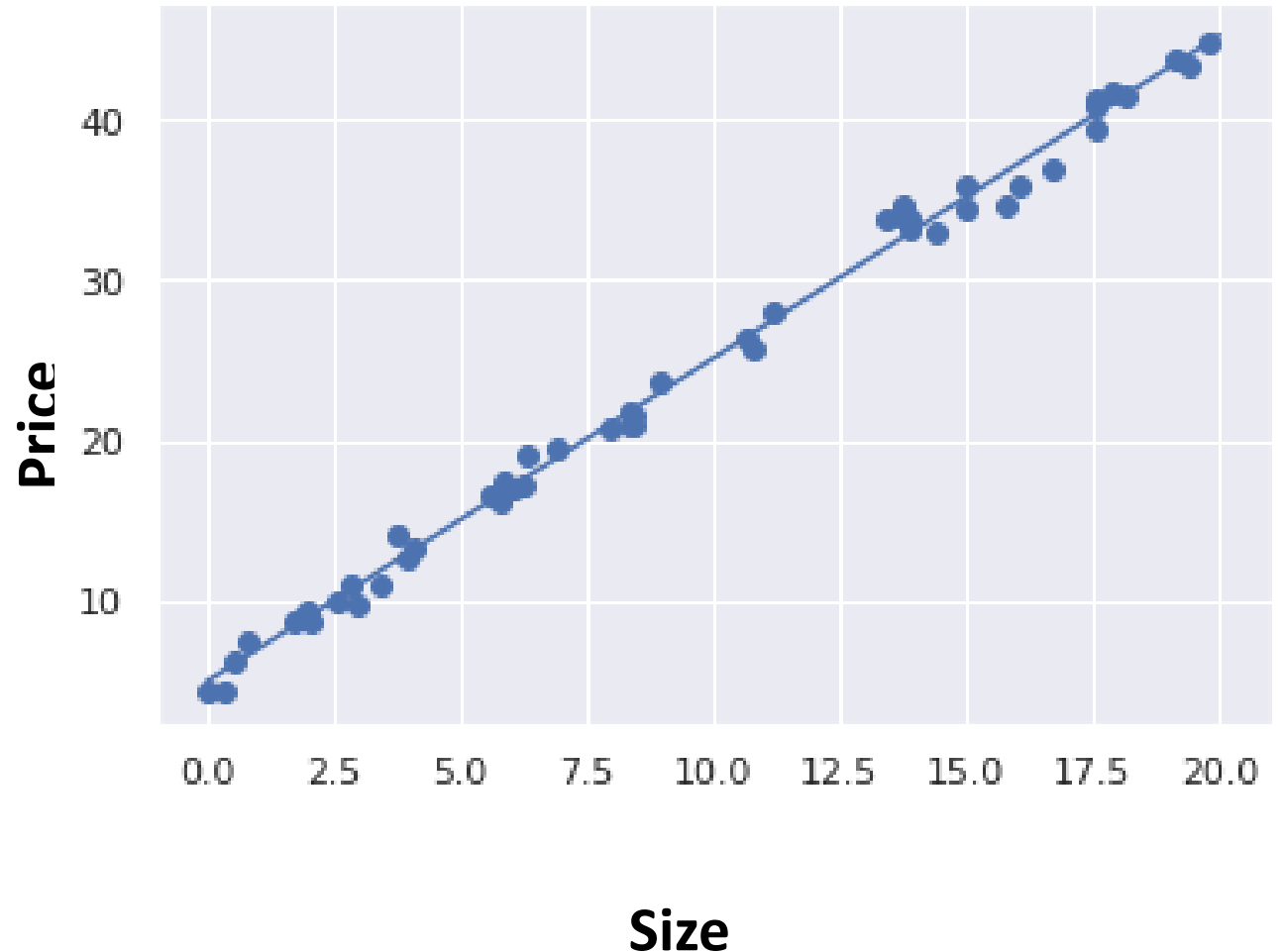
Linear Functions With Varying Values of Θ



Linear Regression with One Variable

- $y' = h\theta(x) = \theta_0 + \theta_1 x$
- Intercept: $\theta_0 = 5$
- Slope: $\theta_1 = 2$

Input Data (x)	Correct Answer (y)
5	15
10	25
15	35
..	..



Cost Function

- **Hypothesis Function**

$$y' = h\vartheta(x) = \vartheta_0 + \vartheta_1 x$$

- **Cost function** (to measure the performance of hypothesis function)

$$J(\vartheta_0, \vartheta_1) = \frac{1}{2m} \sum_{i=1}^m (y'^{(i)} - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^m (h\vartheta(x^{(i)}) - y^{(i)})^2$$

Cost Function

Input Data (x)	Correct Answer (y)	Estimated Answer	Error
8.3	20.99	21.6	-0.61
14.4	32.89	33.8	-0.81
6.05	17.1	17.08	0.02
..

$$\text{Mean Square Error (MSE)} = J(\vartheta_0, \vartheta_1) = \frac{1}{2m} \sum_{i=1}^n (y^{(i)} - y'^{(i)})^2$$

Cost Function

- **Hypothesis Function**

$$y' = h\vartheta(x) = \vartheta_0 + \vartheta_1 x$$

- **Cost function** (to measure the performance of hypothesis function)

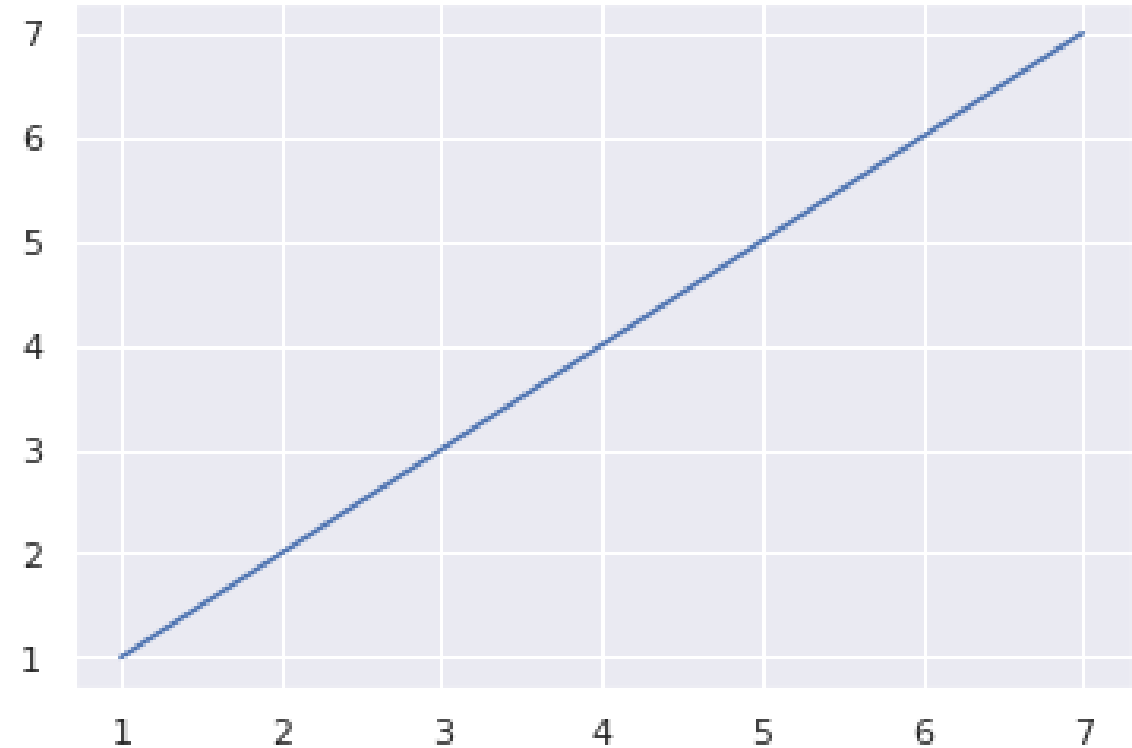
$$J(\vartheta_0, \vartheta_1) = \frac{1}{2m} \sum_{i=1}^m (y'^{(i)} - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^m (h\vartheta(x^{(i)}) - y^{(i)})^2$$

- **Objective:**

$$\min_{\vartheta_0, \vartheta_1} J(\vartheta_0, \vartheta_1)$$

Cost Function - Example

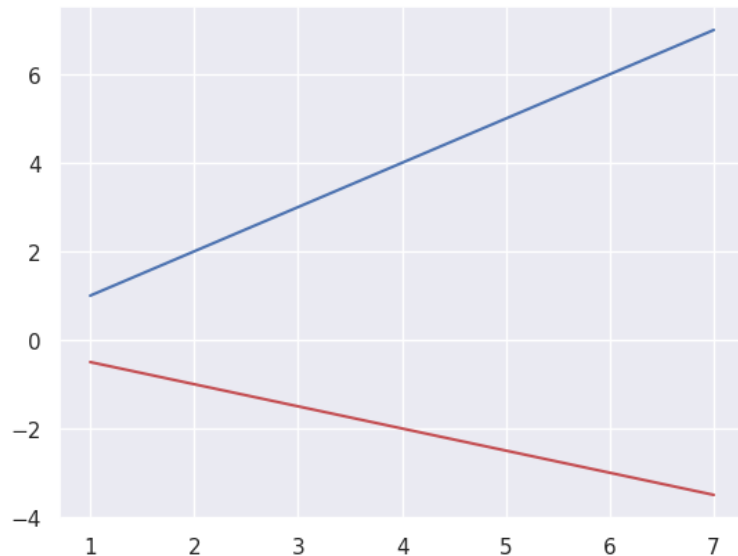
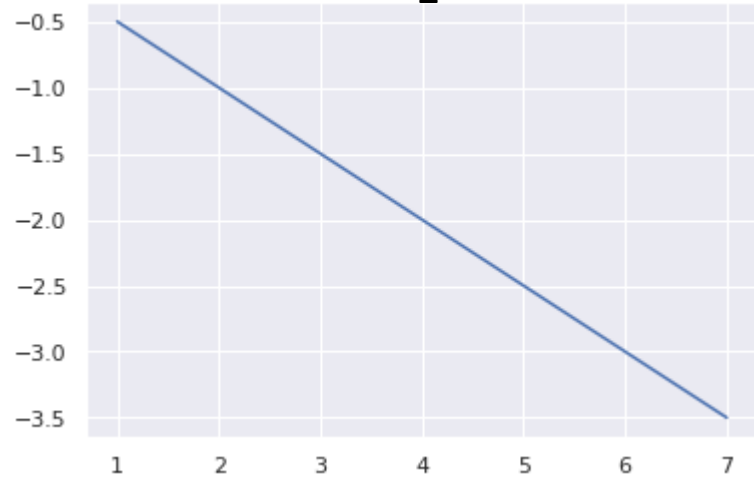
- $\mathbf{x} = [1, 2, 3, 4, 5, 6, 7]$
- $\mathbf{y} = [1, 2, 3, 4, 5, 6, 7]$
- $y' = h\theta(x) = \theta_0 + \theta_1 x$
- Assume $\theta_0 = 0$
- So, $y' = h\theta(x) = \theta_1 x$



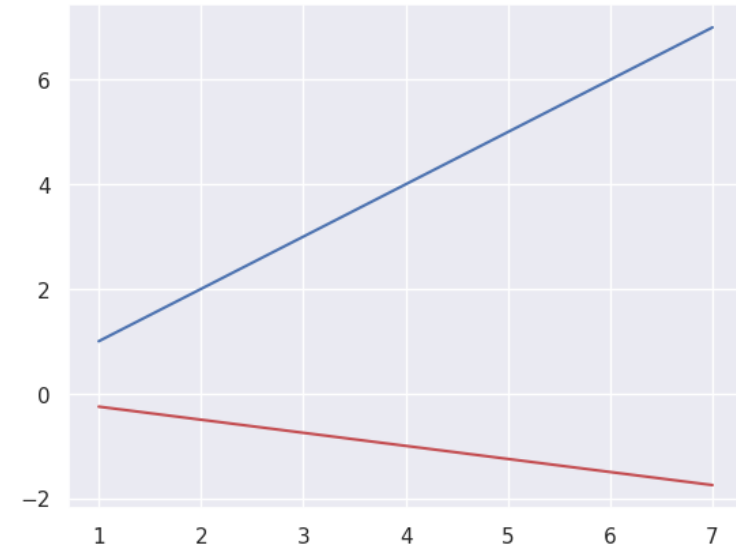
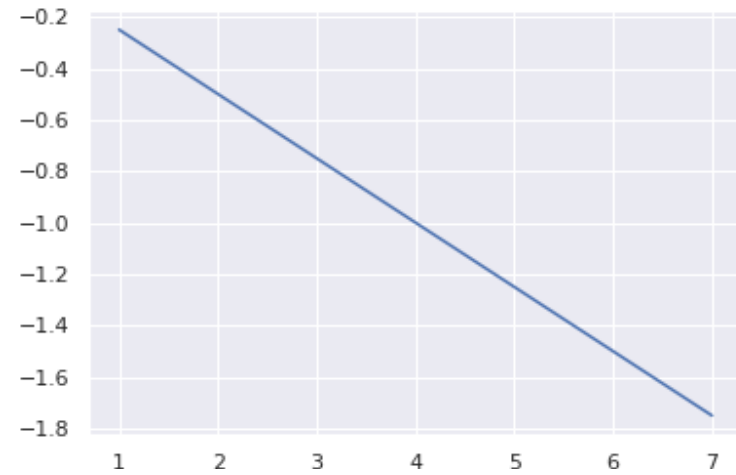
$\theta_1 = [-0.5, -0.25, 0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25]$

Cost Function - Example

- $\vartheta_1 = -0.5, J(\vartheta_1) = 45.0$

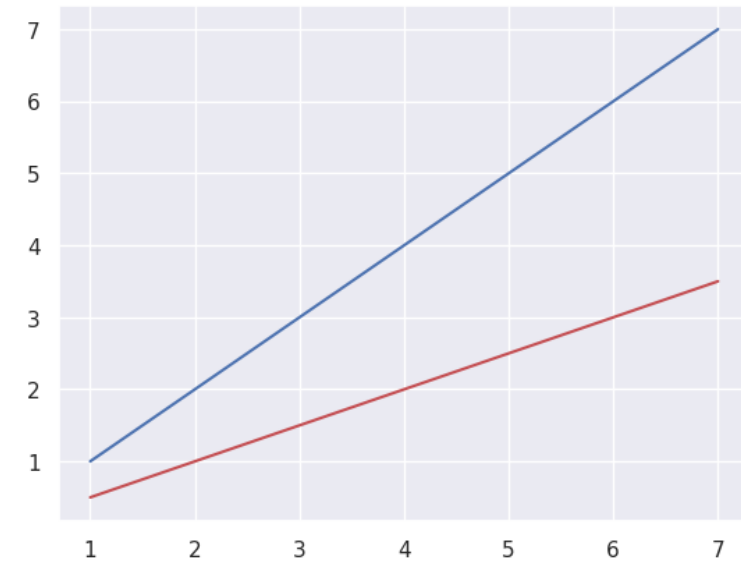
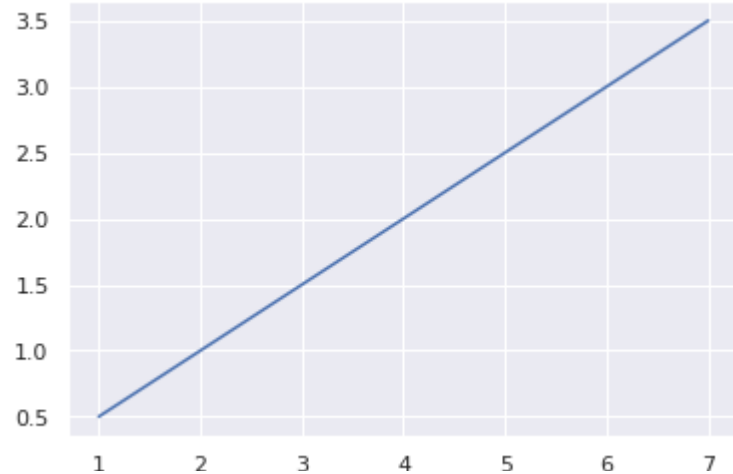


- $\vartheta_1 = -0.25, J(\vartheta_1) = 31.25$

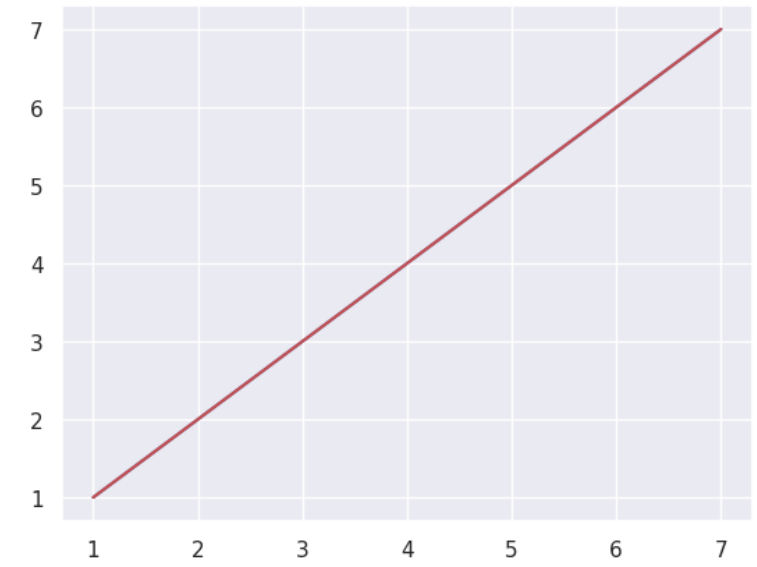
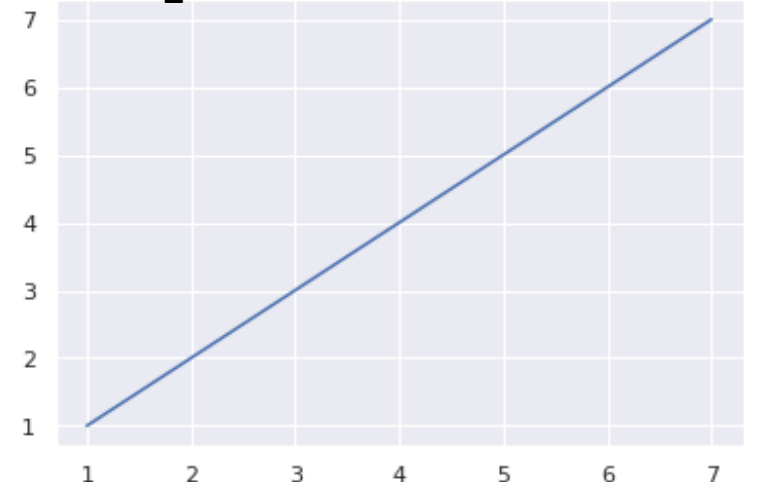


Cost Function - Example

- $\vartheta_1 = 0.5$, $J(\vartheta_1) = 5.0$

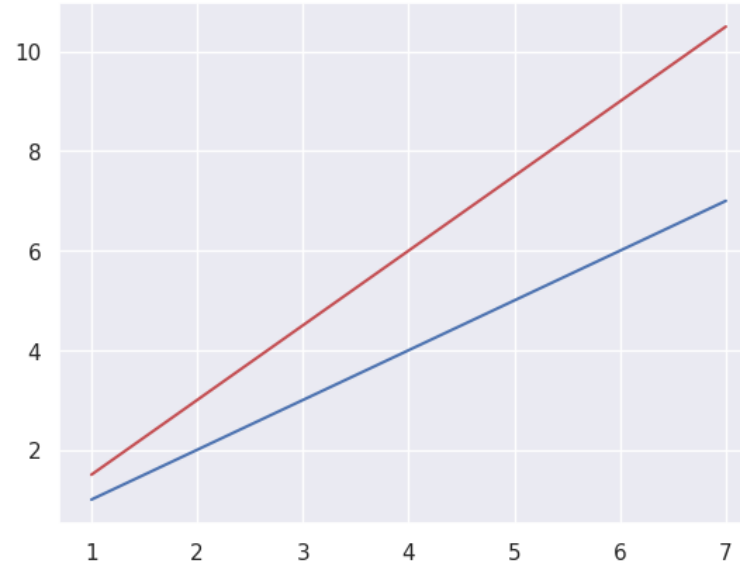


- $\vartheta_1 = 1.0$, $J(\vartheta_1) = 0.0$

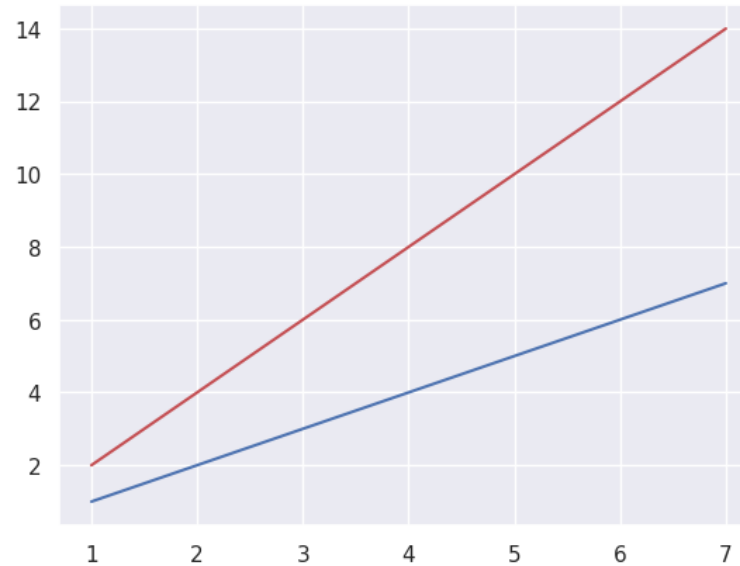


Cost Function - Example

- $\vartheta_1 = 1.5, \quad J(\vartheta_1) = 5.0$

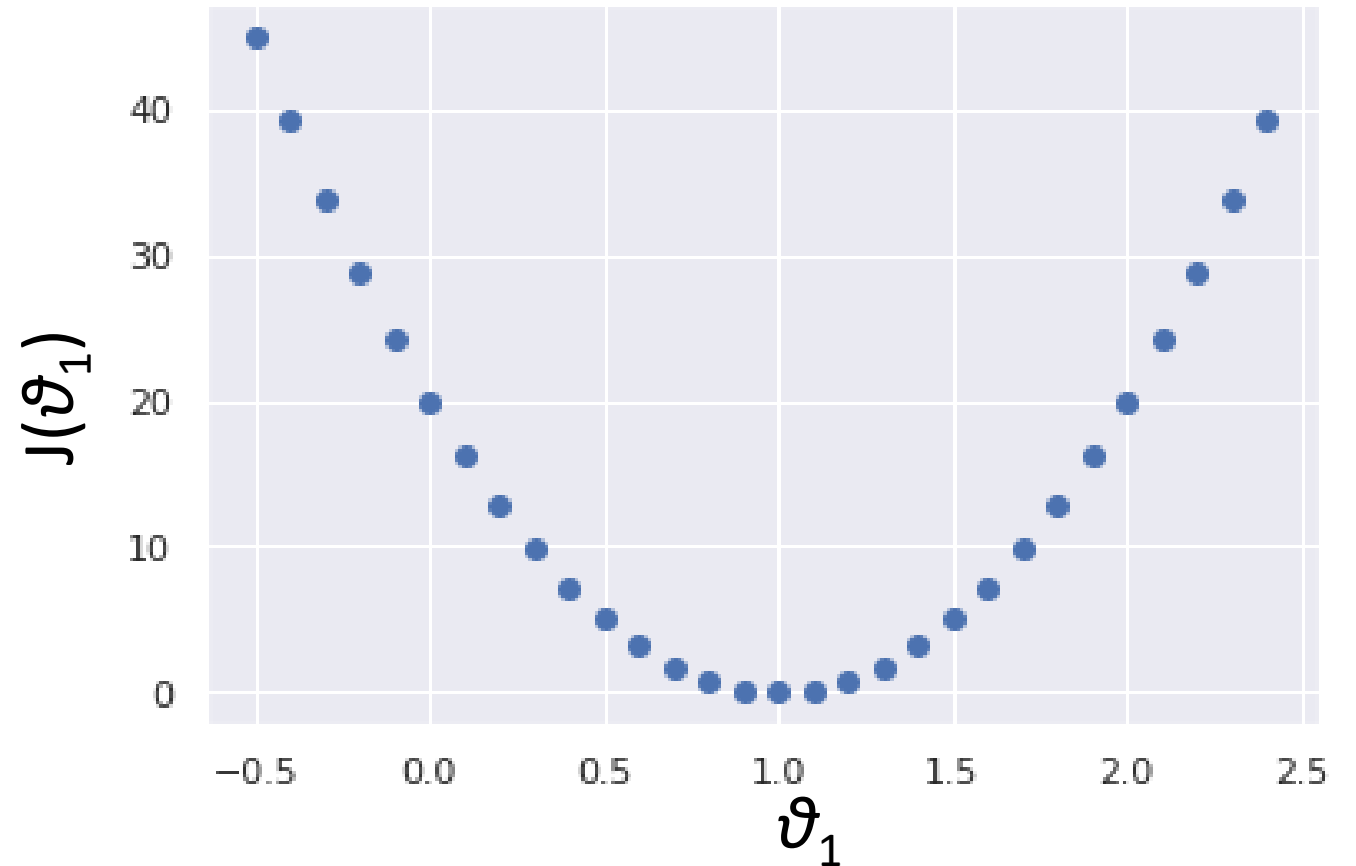


- $\vartheta_1 = 2.0, \quad J(\vartheta_1) = 20.0$

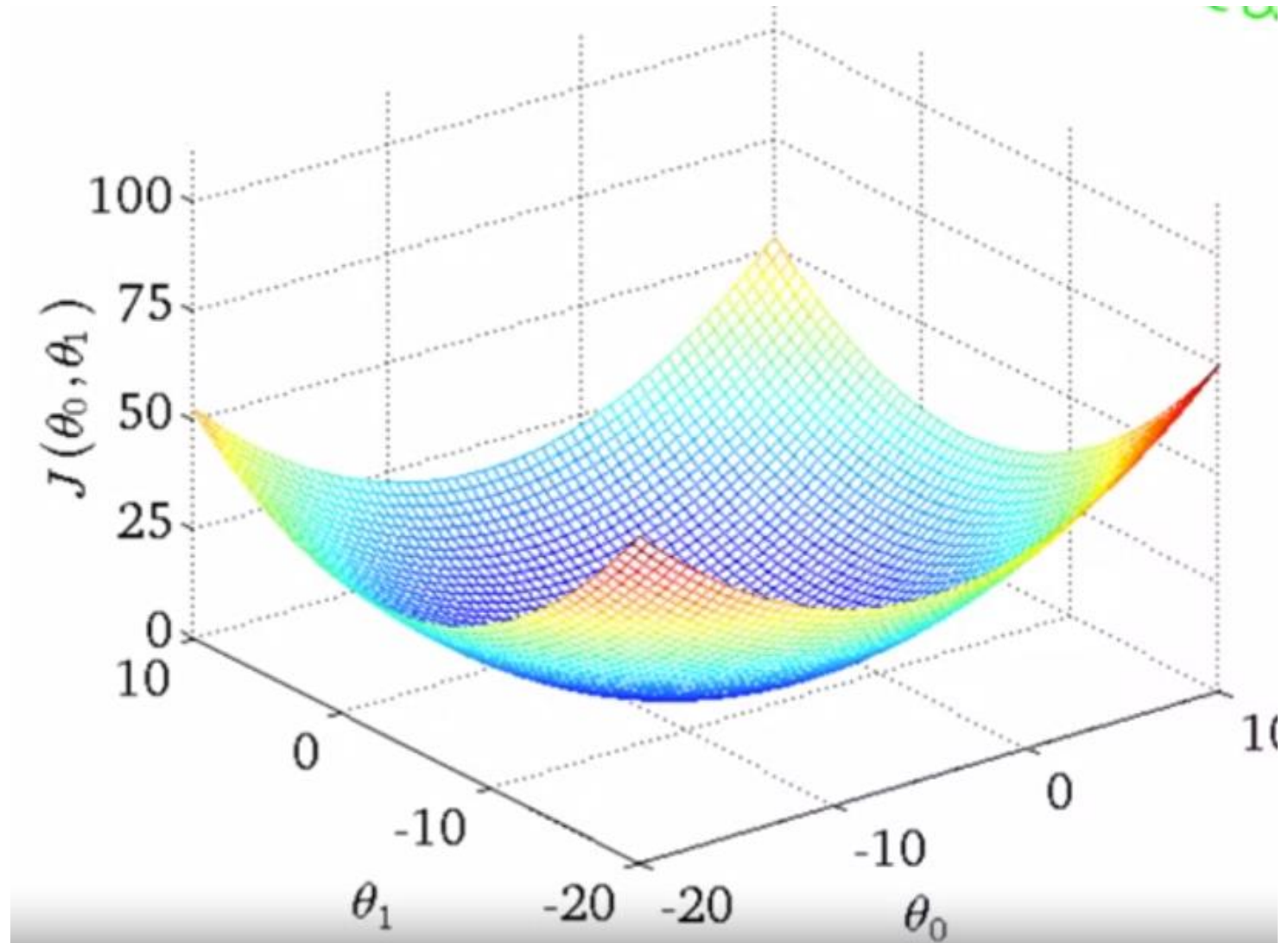


Graph of $J(\vartheta_1)$

- $\theta_1 = [-0.5, -0.25, 0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25]$
- $J(\theta_1) = [45.0, 31.25, 20.0, 11.25, 5.0, 1.25, 0.0, 1.25, 5.0, 11.25, 20.0, 31.25]$



Graph of $J(\vartheta_0, \vartheta_1)$

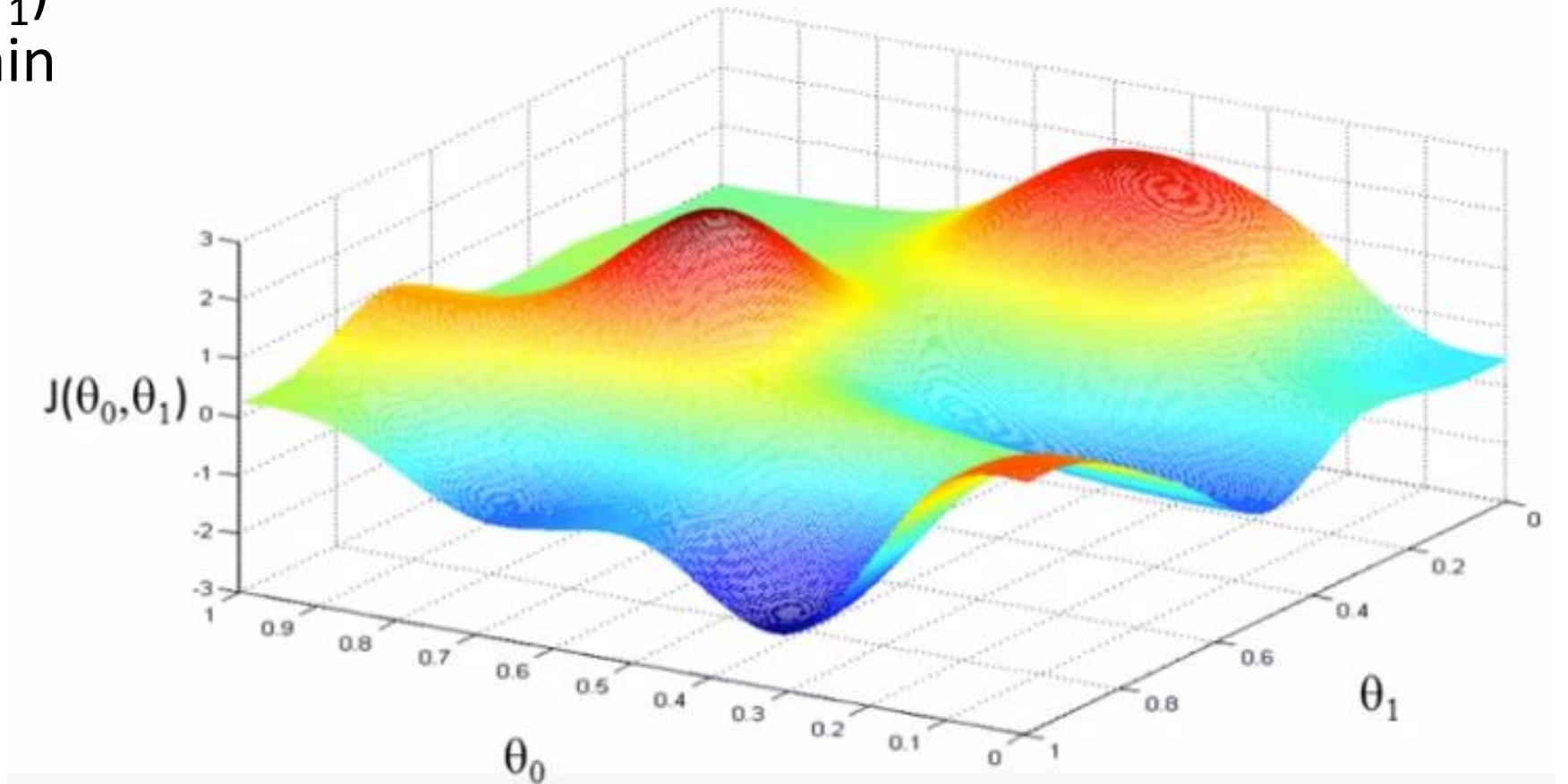


Some Concepts

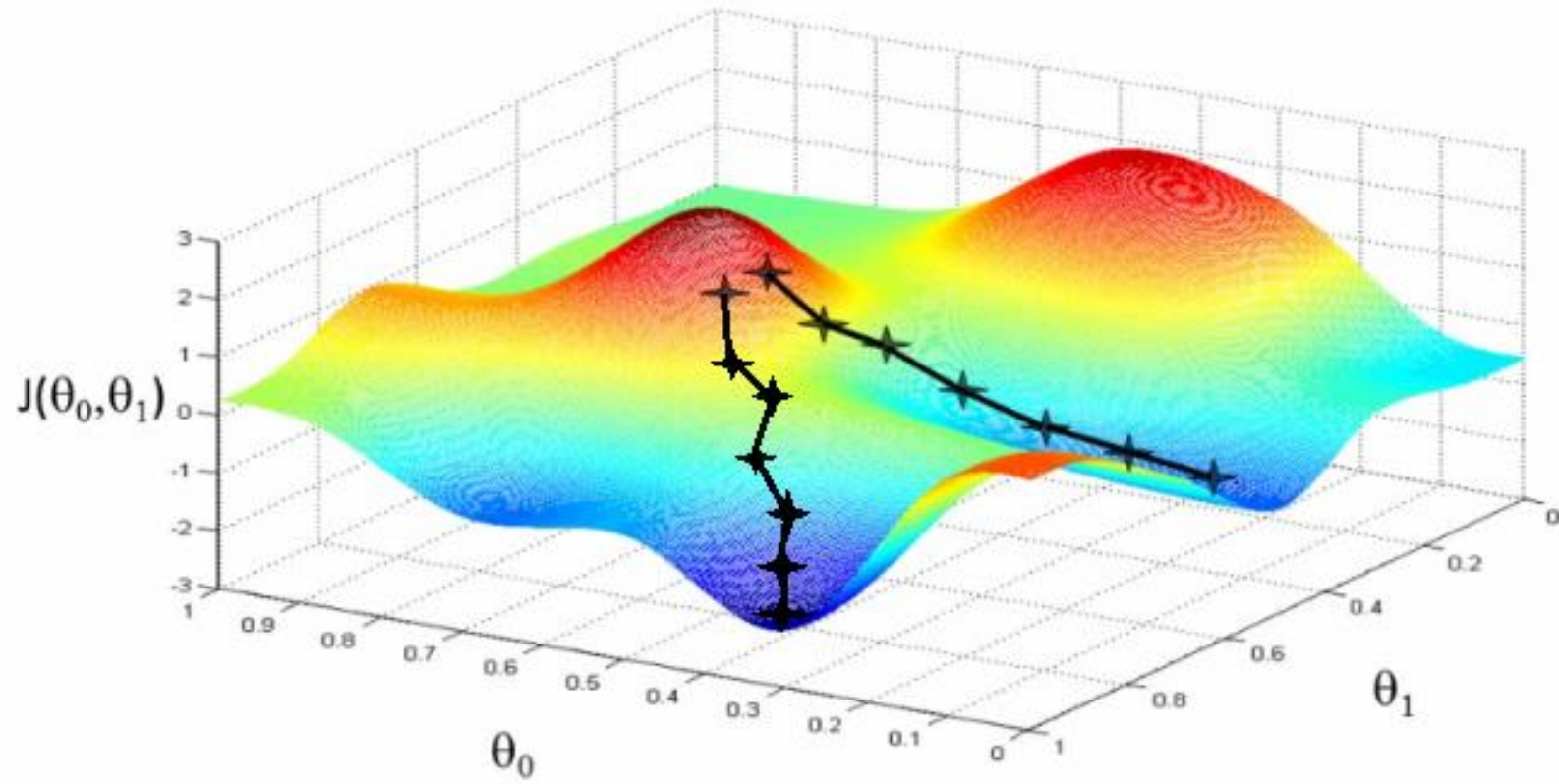
- Global Minimum
- Local Minimum
- Convex Functions

Gradient Descent Algorithm

- We have $J(\vartheta_0, \vartheta_1)$
and we want \min
 $J(\vartheta_0, \vartheta_1)$



Solving Minimization Problem



Recap of the Latest Contents

- Univariate Linear Regression

$$y' = h_{\theta}(x) = \theta_0 + \theta_1 x$$

- Cost/Loss function, Mean Squared Error

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (y'^{(i)} - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Derivatives

$$f(x) = 4x$$

$$f(x) = x^3$$

$$f(x) = (x + 2)^4$$

$$f(x,y) = (3x + 2y + 2)^2$$

Gradient Descent Algorithm

- $\vartheta_j := \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1)$ (for $j = 0$ and $j = 1$)
- $\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1)$ is a partial derivative term
- α : (Alpha) is learning rate
- Simultaneous Update
- $\text{temp0} = \vartheta_0 - \alpha \frac{\partial}{\partial \vartheta_0} J(\vartheta_0, \vartheta_1)$
- $\text{temp1} = \vartheta_1 - \alpha \frac{\partial}{\partial \vartheta_1} J(\vartheta_0, \vartheta_1)$
- $\vartheta_0 := \text{temp0}$
- $\vartheta_1 := \text{temp1}$

Linear Regression with Gradient Descent

$$\vartheta_j := \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1)$$

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \left(\frac{1}{2m} \sum_{i=1}^m (h\vartheta(x^{(i)}) - y^{(i)})^2 \right)$$

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \left(\frac{1}{2m} \sum_{i=1}^m (\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)})^2 \right)$$

Linear Regression with Gradient Descent

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \left(\frac{1}{2m} \sum_{i=1}^m (\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)})^2 \right)$$

$$\frac{\partial}{\partial \vartheta_0} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_0} \left(\frac{1}{2m} \sum_{i=1}^m (\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)})^2 \right)$$

$$\frac{\partial}{\partial \vartheta_0} J(\vartheta_0, \vartheta_1) = \frac{1}{m} \sum_{i=1}^m (\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)})$$

$$\frac{\partial}{\partial \vartheta_1} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_1} \left(\frac{1}{2m} \sum_{i=1}^m (\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)})^2 \right)$$

$$\frac{\partial}{\partial \vartheta_1} J(\vartheta_0, \vartheta_1) = \frac{1}{m} \sum_{i=1}^m ((\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)}) x^{(i)})$$

Linear Regression with Gradient Descent

- Repeat until **converge**

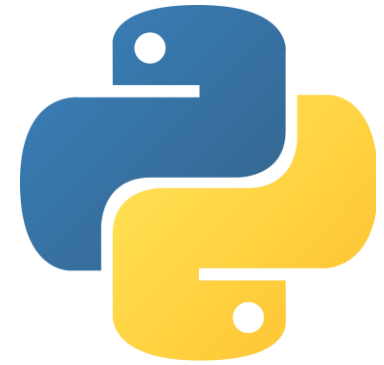
$$\vartheta_0 := \vartheta_0 - \alpha \left(\frac{1}{m} \sum_{i=1}^m (\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)}) \right)$$

$$\vartheta_1 := \vartheta_1 - \alpha \left(\frac{1}{m} \sum_{i=1}^m (\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)}) x^{(i)} \right)$$

- Simultaneous update

What is Python?

- High level programming language
- First released in 1991 by Guido van Rossum
- Object-oriented, imperative, functional and procedural
- Supported by many operating systems
- Commonly used for computation in the field of artificial intelligence and machine learning



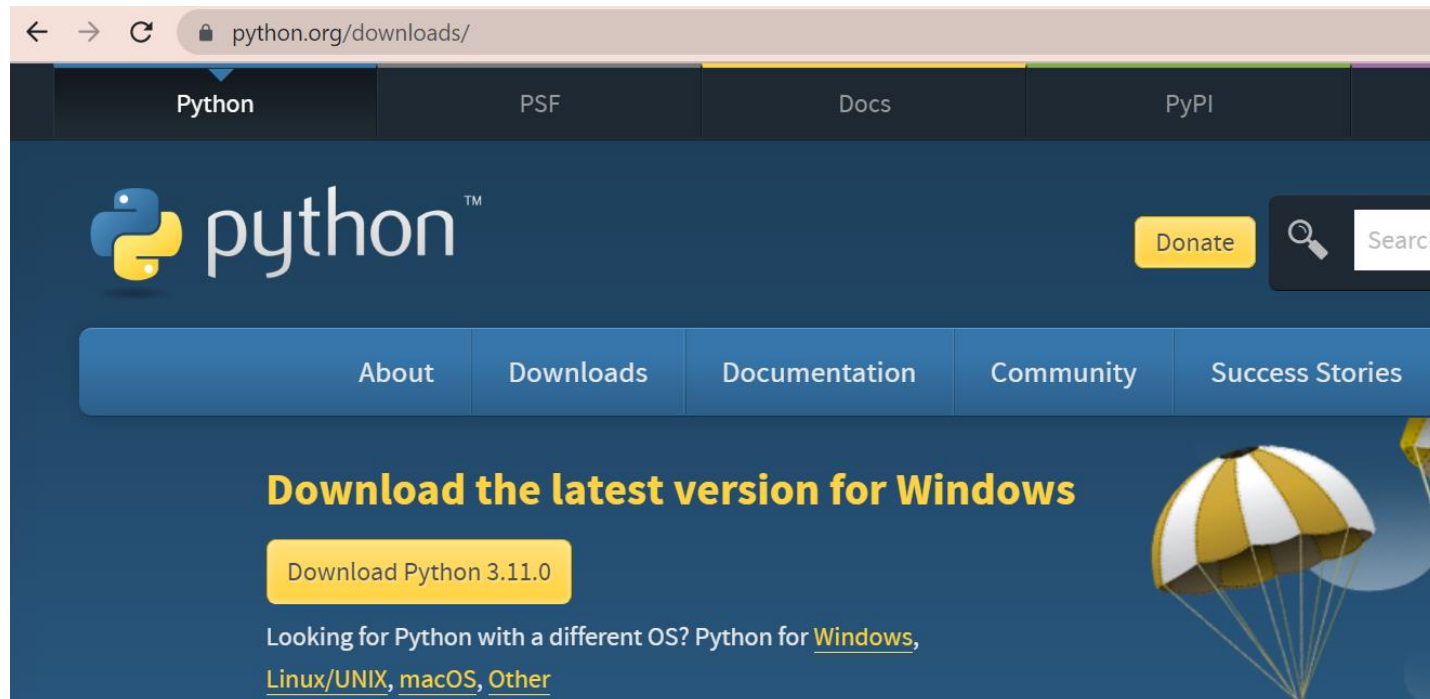
Why Python?

- Wide scientific community:
 - Many different helpful libraries
 - Open source frameworks and tools
 - Python Package Index (PyPI): 433,519
- Easy to learn/use
- Simplicity and consistency,
- High-level language: ideas can be implemented quickly
- Flexibility,
- Platform independence



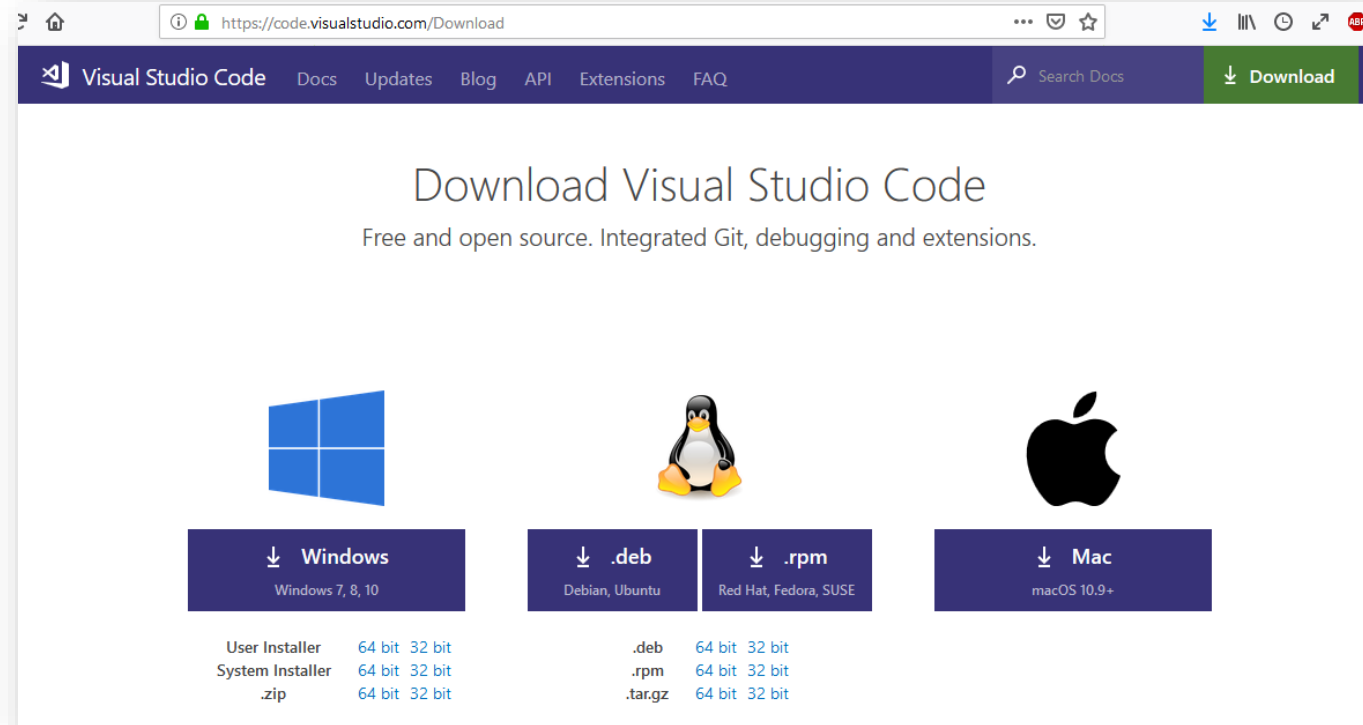
Installation

- **Windows:** visit <https://www.python.org/downloads/> to download Python (recommended version: 3.9 or above)
- **Linux:** run the following command in terminal
sudo apt-get install python3.9



Installation IDE

- The choice of the IDE is up to you. I nevertheless recommend **VS Code** and will provide instructions for this IDE.
 - To download VS Code, visit <https://code.visualstudio.com/Download>



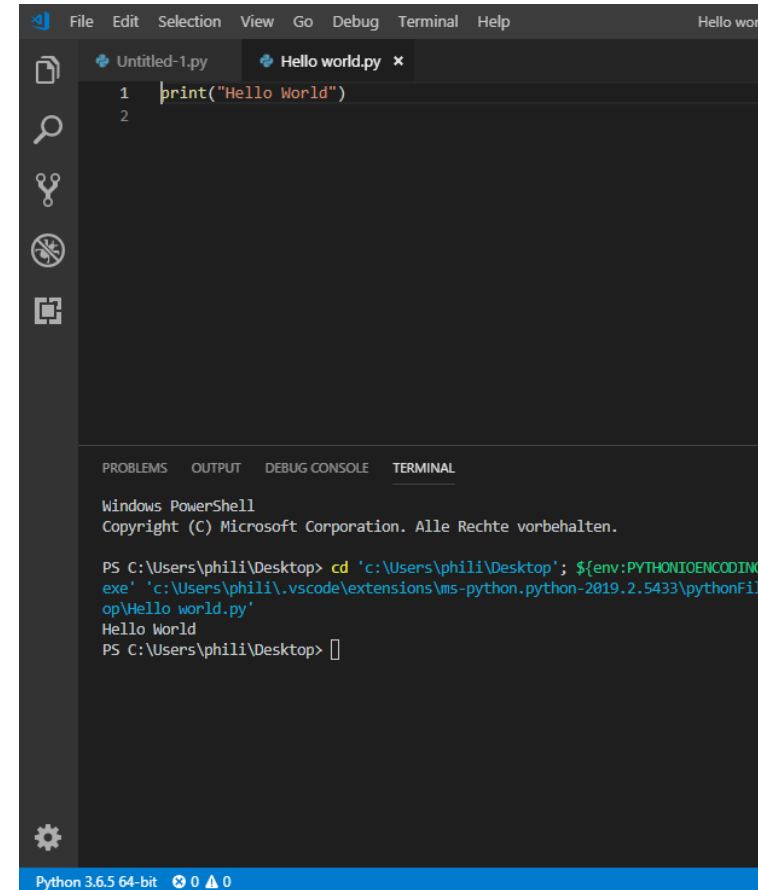
Setup VS Code

- Open the „Extensions“ view by:
 - Selecting „View“ -> „Extensions“
 - CTRL + Shift + X
- Search and install „Python“



VS Code (Python script)

- Create a new file (File - New File)
- Save the file (File-> Save As...)
- Make sure it is of type python (.py)
- Scripts can be executed with F5 (debug mode), or right click + “Run Python file in terminal”
- To check your installation: print “Hello world!”



The screenshot shows the Visual Studio Code interface. The editor window displays a file named 'Hello world.py' with the following code:

```
1 print("Hello World")
2
```

The bottom panel shows the 'TERMINAL' tab, which contains the output of running the script in a Windows PowerShell terminal:

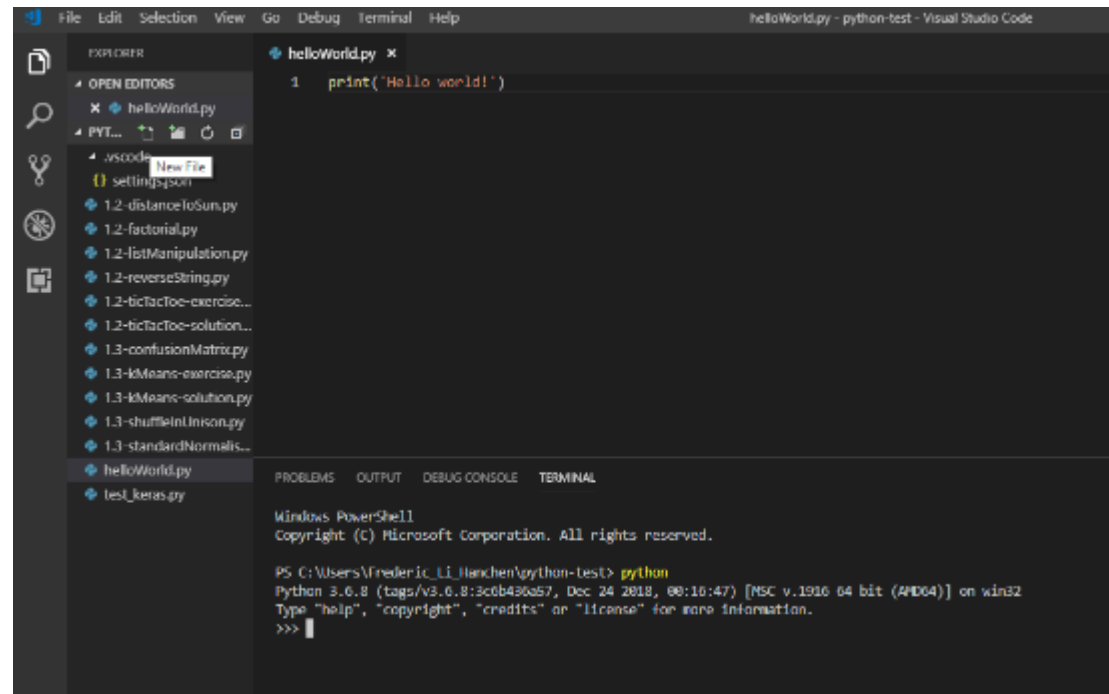
```
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\phili\Desktop> cd 'c:\Users\phili\Desktop'; ${env:PYPATH}
exe' 'c:\Users\phili\.vscode\extensions\ms-python.python-2019.2.5433\pythonFil
op\Hello world.py'
Hello World
PS C:\Users\phili\Desktop>
```

The status bar at the bottom indicates 'Python 3.6.5 64-bit'.

How to run Python code?

- Two main ways to execute Python code:
 1. Start the **Python interpreter** by typing `> python` in command line
 2. Write a **Python script** (.py file) then execute it by typing `> python name_of_the_script.py` in command line



Python Identifiers

- A **Python identifier** is a name used to identify a variable, function, class, module or other object:
 - Starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
 - Case sensitive.
- When using a Python identifier, avoid:
 - Using a Python keyword.
 - Starting or ending the identifier with underscore (_) unless you have a specific reason to do so.
- Python is a **dynamically typed** language:
 - No need to declare any variable type.
 - The type of a variable can be changed in the same script.
 - E.g.: `aVariable = 0`

```
'''  
aVariable = "Hello world!"
```

Python keywords

- Reserved words (cannot use them as constant or variable)

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Naming conventions

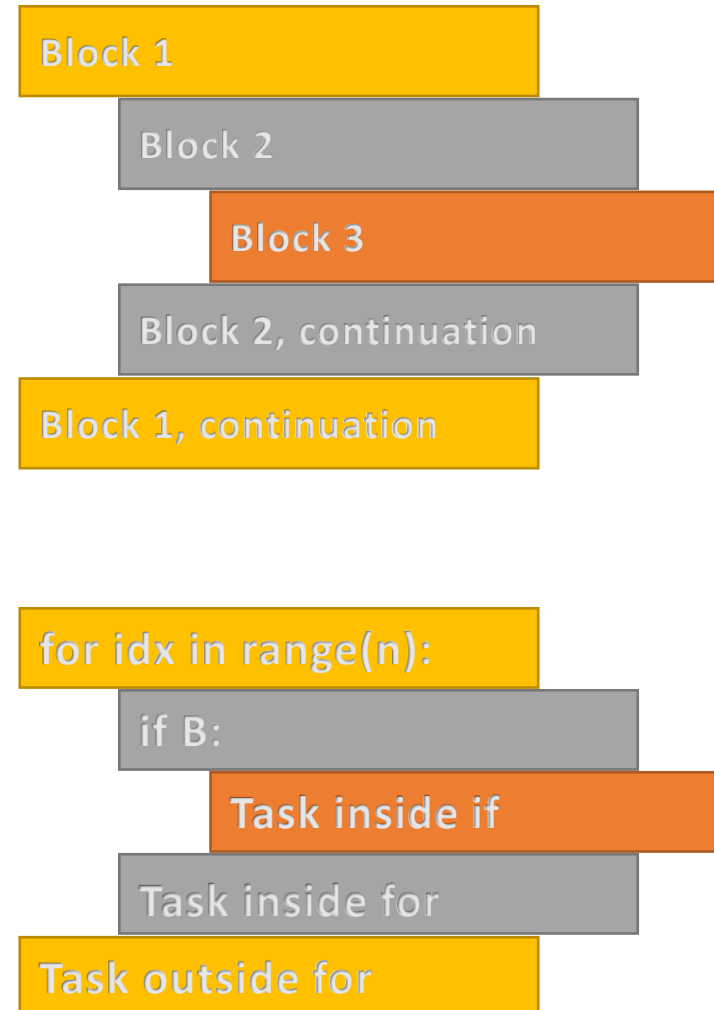
https://visualgit.readthedocs.io/en/latest/pages/naming_convention.html

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- Identifiers which start and end with two trailing underscores are language-defined special names. The most useful examples are:
 - `__init__` which refer to the class constructor
 - `__name__` and `__main__` which are used to define a main file:

```
if __name__ == "__main__":  
    .....
```

Indentation

- In Python, no braces to indicate blocks of code
- Denoted by **line indentation**
- Number of spaces in the indentation up to you, but all statements within the block must be indented the same amount
 - Commonly used indentations: **tabs** or **4 spaces**.
- Notes:
 - mixing spaces and tabs together → error!
 - no need to use any character to denote an end of line (e.g. “;”)



Multi-Line Statements

- Statements in Python typically end with a new line
- Use of the line continuation character (\)

```
total = item_one + \  
        item_two + \  
        item_three
```

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character

```
days = ['Monday', 'Tuesday', 'Wednesday',  
        'Thursday', 'Friday']
```

Quotation

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals
- Start and end same type
- Triple quotes are used to span the string across multiple lines

```
word = 'word'  
sentence = "This is a sentence."  
paragraph = """This is a paragraph. It is  
made up of multiple lines and  
sentences."""
```

Comments

- A **hash sign (#)** that is not inside a string literal begins a comment -> whole line is commented (ignored by the interpreter)
- Can be used to comment multiple lines in a row
- **Triple-quoted string** is also ignored by Python interpreter and can be used as a multiline comments

```
#This is a comment  
print("Hello World")  
  
'''  
This is  
a  
multirow comment  
'''
```

Basic operations

- The operators `+`, `-`, `*` and `/` work just like in most other languages
- The standard comparison operators are written the same as in C:
 - `<` (less than)
 - `>` (greater than)
 - `==` (equal to)
 - `<=` (less than or equal to)
 - `>=` (greater than or equal to)
 - `!=` (not equal to)
- Modulo operation: `%`
- Floor division: `a // b`

Assigning values to variables

- Variables do not need explicit declaration to reserve memory space
- No type needed
- Equal sign (=)
- delete entire variables: *del *variable name**

```
# An integer assignment
counter = 100
# A floating point
miles = 1000.0
# A string
name = "John"
# one line
counter, miles, name = 100, 1000.0, "John"
# Assign single value to several variables
a = b = c = 1
# Delete a variable
del a
```

Standard data types

- Python has six standard data types
 - Numbers
 - Boolean
 - String
 - List
 - Tuple
 - Dictionary
- For numbers: integer, float, complex
- For Booleans: `True`, `False`
 - Note: in Python 3.x, Booleans inherit from integers, i.e. `True == 1` and `False == 0` will both return `True`.

Lists

- Items separated by commas and enclosed within square brackets ([])
 - Similar to arrays in C.
- **Indexing starts at 0**
- Items can be of different data type
- Items can be deleted: *del*

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list) # Prints complete list
print(list[0]) # Prints first element of the list
print(list[1:3]) # Prints elements starting from 2nd till 3rd
print(list[2:]) # Prints elements starting from 3rd element
print(tinylist * 2) # Prints list two times
print(list + tinylist) # Prints concatenated lists
```

IF statement

```
x = 1
if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

- Form
if *condition* :
 code block
- Elif/else are optional

Loops - For

- Iterates over the items of any sequence
- Iterate over a slice copy of the entire list with *list[:]*

```
list = ['element1', 'element2', 'element3']  
for element in list:  
    print(element)
```

- Iterate over a sequence of numbers: *range* - function

```
for i in range(5):  
    print(i)
```



0
1
2
3
4

* *range()* returns an iterator, not a list!

Loops - While

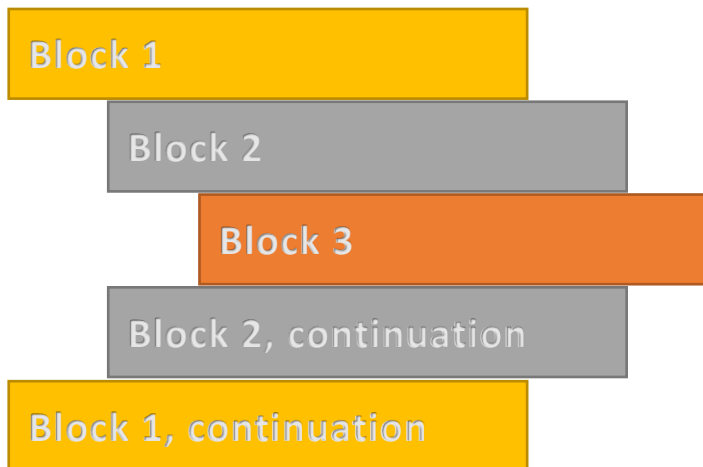
- The while statement is used for repeated execution as long as an expression is true
- Optional else clause: will be executed the first time the expression is false and the loop terminates

```
i = 0
while i < 5:
    print(i)
    i += 1
else:
    print("Finished")
```

- The *break* statement, like in C, breaks out of the innermost enclosing *for* or *while* loop.

Conclusion

- Basic informal introduction to Python
- First basic Python exercises
- Next week: Data Structures and Functions



Exercise 1

- Write a program `printNegative` which prints all negative elements of an integer or float input list.
- Write a program `filterOdd` which takes a list of integers as input and returns a sub-list of the input containing only its odd elements (tip: the Python modulo operator is `%`; e.g. `7%2` returns `1`).
- Write a program `removeDuplicate` which takes a list of any type as input, and returns a list without any duplicate elements as output.

Exercise 2

Write a function `printReverseString` which takes a sentence under the string format as input (e.g. "Hello world!") and prints the sentence with words in the reverse order (e.g. "world! Hello").

Tips:

- A sentence is an ensemble of words separated by spaces
- The Python string methods `split` and `join` should be useful for this exercise

Exercise 3

Write a function `distanceToSun` which asks the user to input the name of a planet of our solar system in command line, and prints the distance between the planet and the Sun in kilometres.

Tips:

- Pick the online source of your choice to find distances between planets and the Sun.
- Asking for an user input in terminal can be done using the `userInput = input("text to display")` syntax.
- Remember to treat the case where the user input is invalid.
- Stopping a function can be done using `return`

What is Python?

- High level programming language
- First released in 1991 by Guido van Rossum
- Object-oriented, imperative, functional and procedural
- Supported by many operating systems
- Commonly used for computation in the field of artificial intelligence and machine learning



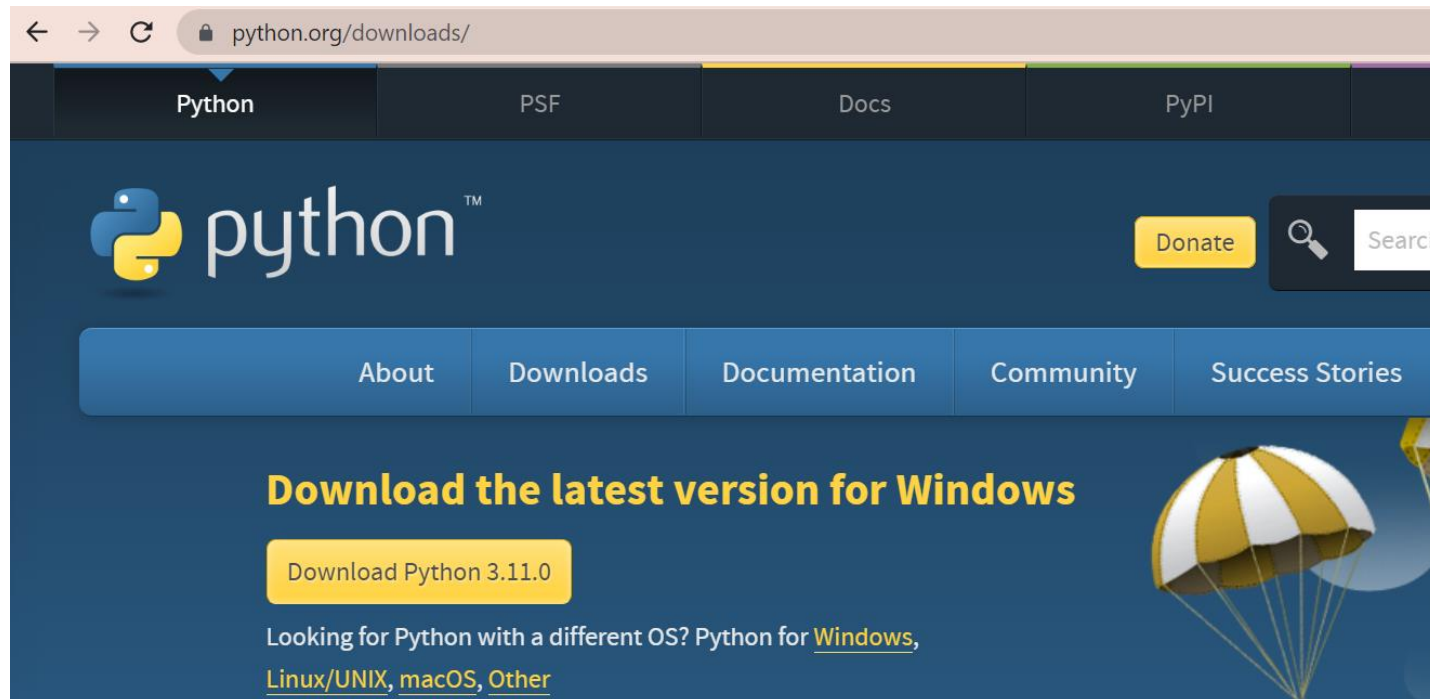
Why Python?

- Wide scientific community:
 - Many different helpful libraries
 - Open source frameworks and tools
 - Python Package Index (PyPI): 433,519
- Easy to learn/use
- Simplicity and consistency,
- High-level language: ideas can be implemented quickly
- Flexibility,
- Platform independence



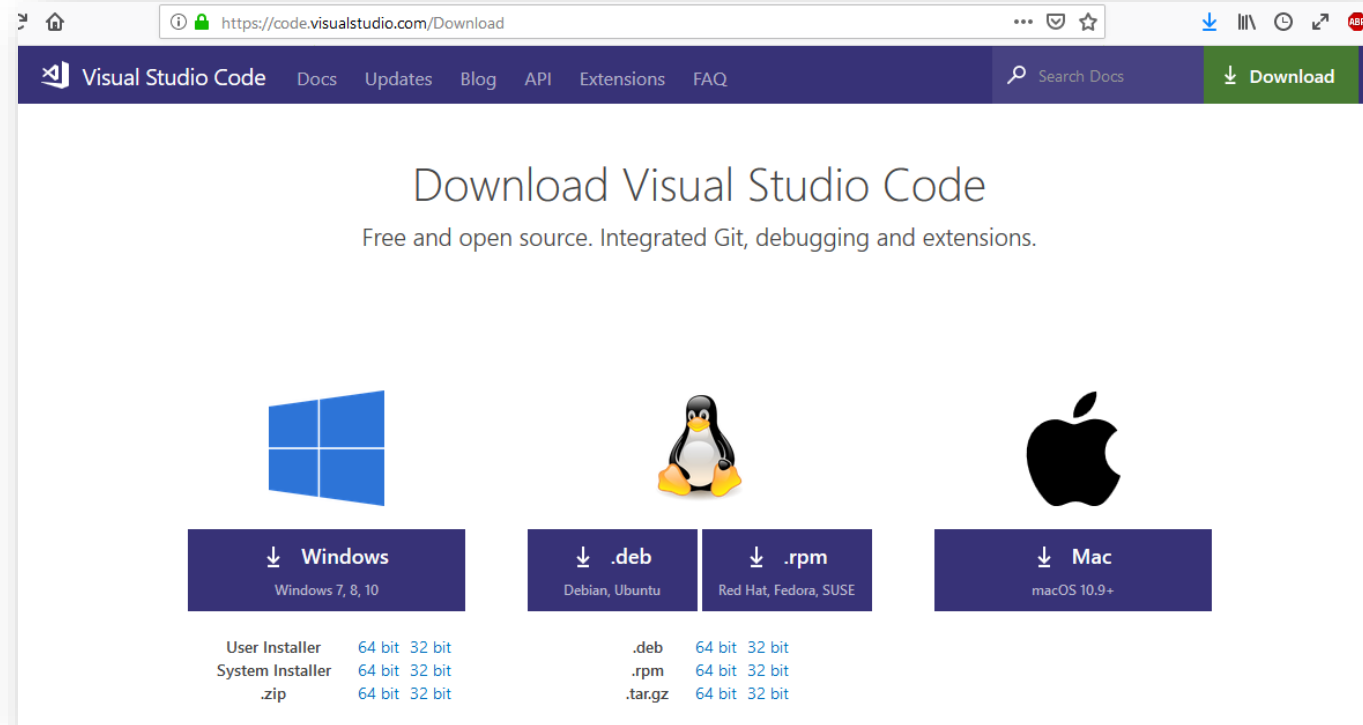
Installation

- **Windows:** visit <https://www.python.org/downloads/> to download Python (recommended version: 3.9 or above)
- **Linux:** run the following command in terminal
sudo apt-get install python3.9



Installation IDE

- The choice of the IDE is up to you. I nevertheless recommend **VS Code** and will provide instructions for this IDE.
 - To download VS Code, visit <https://code.visualstudio.com/Download>



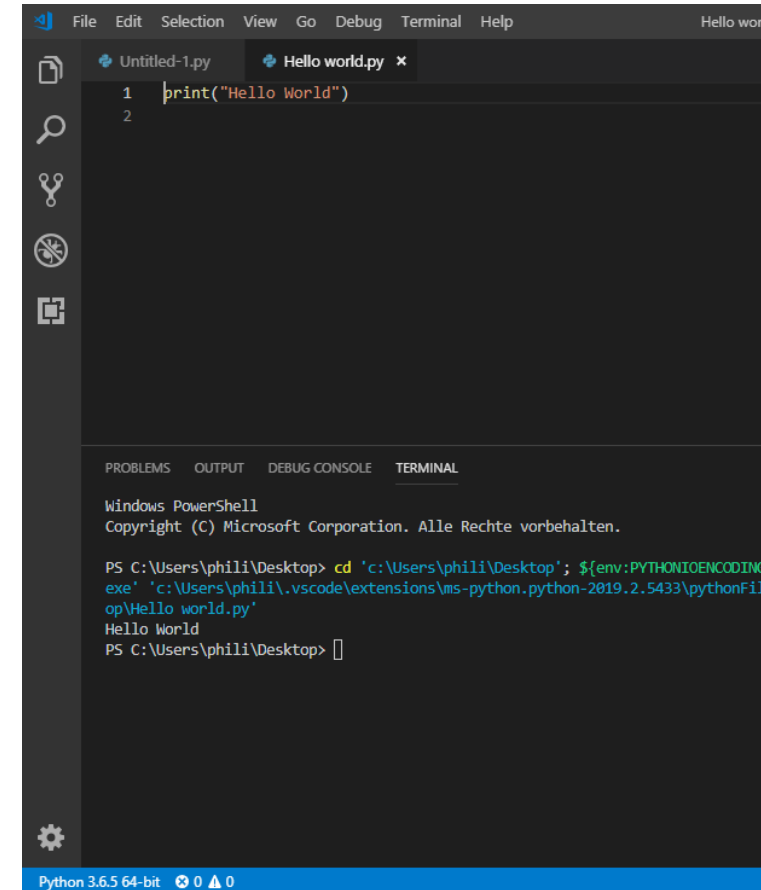
Setup VS Code

- Open the „Extensions“ view by:
 - Selecting „View“ -> „Extensions“
 - CTRL + Shift + X
- Search and install „Python“



VS Code (Python script)

- Create a new file (File - New File)
- Save the file (File-> Save As...)
- Make sure it is of type python (.py)
- Scripts can be executed with F5 (debug mode), or right click + “Run Python file in terminal”
- To check your installation: print “Hello world!”



The screenshot shows the Visual Studio Code interface. The editor window displays a file named 'Hello world.py' with the following code:

```
1 print("Hello World")
2
```

The bottom panel shows the 'TERMINAL' tab, which contains the output of running the script in a Windows PowerShell terminal:

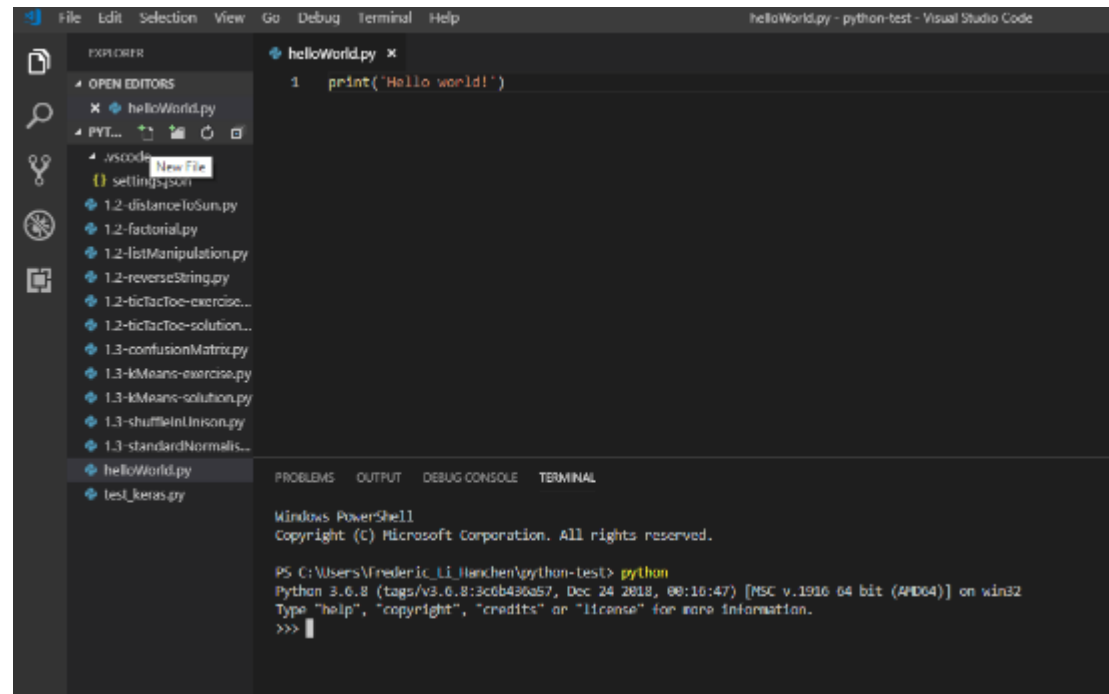
```
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\phili\Desktop> cd 'c:\Users\phili\Desktop'; ${env:PYPATH}
exe' 'c:\Users\phili\.vscode\extensions\ms-python.python-2019.2.5433\pythonFil
op\Hello world.py'
Hello World
PS C:\Users\phili\Desktop>
```

The status bar at the bottom indicates 'Python 3.6.5 64-bit'.

How to run Python code?

- Two main ways to execute Python code:
 1. Start the **Python interpreter** by typing `> python` in command line
 2. Write a **Python script** (.py file) then execute it by typing `> python name_of_the_script.py` in command line



Python Identifiers

- A **Python identifier** is a name used to identify a variable, function, class, module or other object:
 - Starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
 - Case sensitive.
- When using a Python identifier, avoid:
 - Using a Python keyword.
 - Starting or ending the identifier with underscore (_) unless you have a specific reason to do so.
- Python is a **dynamically typed** language:
 - No need to declare any variable type.
 - The type of a variable can be changed in the same script.
 - E.g.: `aVariable = 0`
`aVariable = "Hello world!"`

Python keywords

- Reserved words (cannot use them as constant or variable)

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Naming conventions

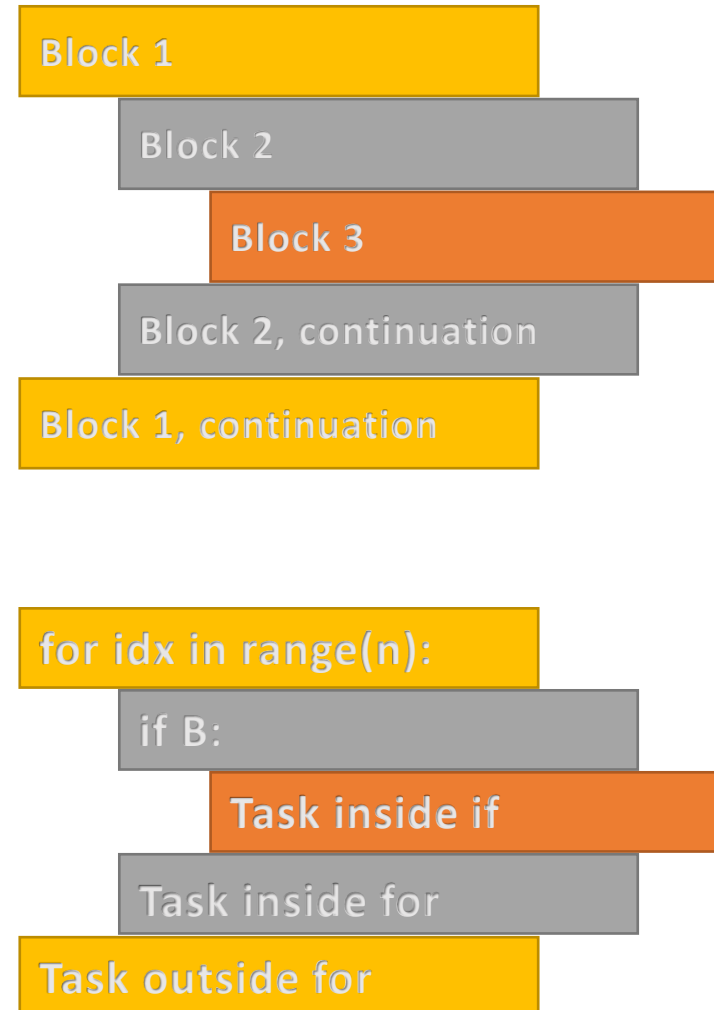
https://visualgit.readthedocs.io/en/latest/pages/naming_convention.html

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- Identifiers which start and end with two trailing underscores are language-defined special names. The most useful examples are:
 - `__init__` which refer to the class constructor
 - `__name__` and `__main__` which are used to define a main file:

```
if __name__ == "__main__":  
    .....
```

Indentation

- In Python, no braces to indicate blocks of code
- Denoted by **line indentation**
- Number of spaces in the indentation up to you, but all statements within the block must be indented the same amount
 - Commonly used indentations: **tabs** or **4 spaces**.
- Notes:
 - mixing spaces and tabs together → error!
 - no need to use any character to denote an end of line (e.g. “;”)



Multi-Line Statements

- Statements in Python typically end with a new line
- Use of the line continuation character (\)

```
total = item_one + \  
    item_two + \  
    item_three
```

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character

```
days = ['Monday', 'Tuesday', 'Wednesday',  
        'Thursday', 'Friday']
```

Quotation

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals
- Start and end same type
- Triple quotes are used to span the string across multiple lines

```
word = 'word'  
sentence = "This is a sentence."  
paragraph = """This is a paragraph. It is  
made up of multiple lines and  
sentences."""
```

Comments

- A **hash sign (#)** that is not inside a string literal begins a comment -> whole line is commented (ignored by the interpreter)
- Can be used to comment multiple lines in a row
- **Triple-quoted string** is also ignored by Python interpreter and can be used as a multiline comments

```
#This is a comment  
print("Hello World")  
  
'''  
This is  
a  
multirow comment  
'''
```


Basic operations

- The operators `+`, `-`, `*` and `/` work just like in most other languages
- The standard comparison operators are written the same as in C:
 - `<` (less than)
 - `>` (greater than)
 - `==` (equal to)
 - `<=` (less than or equal to)
 - `>=` (greater than or equal to)
 - `!=` (not equal to)
- Modulo operation: `%`
- Floor division: `a // b`

Assigning values to variables

- Variables do not need explicit declaration to reserve memory space
- No type needed
- Equal sign (=)
- delete entire variables: *del *variable name**

```
# An integer assignment
counter = 100
# A floating point
miles = 1000.0
# A string
name = "John"
# one line
counter, miles, name = 100, 1000.0, "John"
# Assign single value to several variables
a = b = c = 1
# Delete a variable
del a
```

Standard data types

- Python has six standard data types
 - Numbers
 - Boolean
 - String
 - List
 - Tuple
 - Dictionary
- For numbers: integer, float, complex
- For Booleans: True, False
 - Note: in Python 3.x, Booleans inherit from integers, i.e. `True == 1` and `False == 0` will both return True.

Lists

- Items separated by commas and enclosed within square brackets ([])
 - Similar to arrays in C.
- **Indexing starts at 0**
- Items can be of different data type
- Items can be deleted: *del*

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list) # Prints complete list
print(list[0]) # Prints first element of the list
print(list[1:3]) # Prints elements starting from 2nd till 3rd
print(list[2:]) # Prints elements starting from 3rd element
print(tinylist * 2) # Prints list two times
print(list + tinylist) # Prints concatenated lists
```

IF statement

```
x = 1
if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

- Form
if *condition* :
 code block
- Elif/else are optional

Loops - For

- Iterates over the items of any sequence
- Iterate over a slice copy of the entire list with *list[:]*

```
list = ['element1', 'element2', 'element3']  
for element in list:  
    print(element)
```

- Iterate over a sequence of numbers: *range* - function

```
for i in range(5):  
    print(i)
```



0
1
2
3
4

* *range()* returns an iterator, not a list!

Loops - While

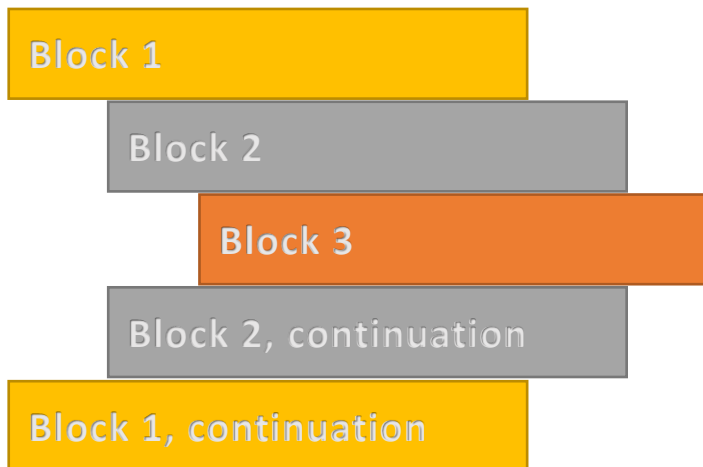
- The while statement is used for repeated execution as long as an expression is true
- Optional else clause: will be executed the first time the expression is false and the loop terminates

```
i = 0
while i < 5:
    print(i)
    i += 1
else:
    print("Finished")
```

- The *break* statement, like in C, breaks out of the innermost enclosing *for* or *while* loop.

Conclusion

- Basic informal introduction to Python
- First basic Python exercises
- Next week: Data Structures and Functions



Exercise 1

- Write a program `printNegative` which prints all negative elements of an integer or float input list.
- Write a program `filterOdd` which takes a list of integers as input and returns a sub-list of the input containing only its odd elements (tip: the Python modulo operator is `%`; e.g. `7%2` returns `1`).
- Write a program `removeDuplicate` which takes a list of any type as input, and returns a list without any duplicate elements as output.

Exercise 2

Write a function `printReverseString` which takes a sentence under the string format as input (e.g. "Hello world!") and prints the sentence with words in the reverse order (e.g. "world! Hello").

Tips:

- A sentence is an ensemble of words separated by spaces
- The Python string methods `split` and `join` should be useful for this exercise

Exercise 3

Write a function `distanceToSun` which asks the user to input the name of a planet of our solar system in command line, and prints the distance between the planet and the Sun in kilometres.

Tips:

- Pick the online source of your choice to find distances between planets and the Sun.
- Asking for an user input in terminal can be done using the `userInput = input("text to display")` syntax.
- Remember to treat the case where the user input is invalid.
- Stopping a function can be done using `return`

Programming - Homework

- Write a Python program that implements a function **computeCost** to compute the Mean Squared Error (MSE) for univariate linear regression. The function should take the following parameters:
 - **X**: A list of m data points (a list of m real numbers).
 - **Y**: A list of m target values (a list of m real numbers corresponding to X).
 - **bias**: The intercept parameter (a real number).
 - **weight**: The slope parameter (a real number).
- The function should return the cost, computed using the MSE formula.

Programming - Homework

- Write a Python program that calculates the cost for different combinations of parameters and prints the minimum cost along with the optimal parameters. The program should include a function `findMinCost` that takes the following parameters:
 - **X**: A list of m data points (a list of m real numbers).
 - **Y**: A list of m target values (a list of m real numbers corresponding to X).
 - **bias_lst**: A list of k possible values for the intercept parameter (a list of k real numbers).
 - **weight_lst**: A list of j possible values for the slope parameter (a list of j real numbers).
- The function should compute the cost for each combination of intercept (from `bias_lst`) and slope (from `weight_lst`), and return the minimum cost and the corresponding parameters.

Reading - Homework

- Machine Learning
 - Resource R1
 - Book B1: 1.3, 2.1, 2.6, 3.1
 - Book B2: Chapter-1, 2
 - Book B3: 2.6