KU x depa
MACHINE LEARNING
Training Course 2022

PYTHON CRASH COURSE

for Data Science and Machine Learning using Google Colab

DR. SETHAVIDH GERTPHOL

KU KASETSART UNIVERSITY depa

Today's Outline

- Using Google Colab
- Jupyter Notebook basics
- Python Crash Course
- Variable and data type
- Expression and Assignment
- Function
- Selection and Loop
- Object and method
- List/String and Iterator
- Module

Google Colab

- Online environment (virtual machine, software, libraries) for programming and data science
- Based on Jupyter Notebook
- Accessed at https://colab.research.google.com/
  - Must have a google account (free)
- Has many data science and machine learning libraries built-in
- Can be used for free and pay for more resources

Jupyter Notebook

- A webapp frontend for IPython
- IPython is an interactive version of Python that allows line-by-line code execution and OS interaction
- Can insert other elements such as text, markdown, images, and link into the same document with code
- Can connect to several kernels such as IPython, R, C, Scala, etc

Welcome To Colaboratory

File Edit View Insert Runtime Tools Help Cannot save changes

Share

Table of contents

+ Code + Text Copy to Drive

RAM Disk Editing

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[1] seconds_in_a_day = 24 * 60 * 60
    seconds_in_a_day
```

```
86400
```

Jupyter Notebook Basics

- Cell: a block of elements (code, text, markdown, etc)
- Cell types
  - Code
  - Markdown (text)
  - Raw
- Operations
  - Add, Delete, Move cell
  - Copy and Paste cell
  - Run cell
  - Many keyboard shortcuts for these operations

Markdown basics

- Markdown: a format that allows quick formatting of text as HTML
- Works by adding symbols to signify formatting
- Symbols
  - `# text` : Header Level up to `###### (level 6)`
  - `**text**` : bold
  - `*text*` or `_text_` : italics
  - `` `text` `` : monospace
  - `-` or `*` : bulleted list
  - `1.` : numbered list
  - `[text](link)` : an html link
  - `![text](link)` : image

[Markdown Guide - Colaboratory (google.com)]

Python Crash (and incomplete) Course

- Data Type and Variable

- Expression
- Function
- Selection and Loop
- Object and method
- List and String
- Dictionary
- Module

Python Crash Course

Using _ as a thousand separator

Scientific notation

- Data Type:
  - Integer: `0, 4, -56, 10_000`
  - Float: `2.4, 0.0, -567.14, 1E5, 10_000.01`
  - String: `"Hello", 'Hello'`
    - Must enclose string with single quote (`'`) or double quote (`"`)
    - Python has no data type for a single character
  - Boolean: True, False
- Literals: things that has the same value with their appearance
  - All of the above are literals
- IPython will output a literal when running a cell with that literal
  - In regular python, must use a print statement

Variables

- Variables: container that can be assigned a value
- Variable names: name used to refer to a variable
- Rules for variable name
  - Can use both upper- and lower-case English characters
    - Name is case sensitive
  - Can use number, but not as the first character
  - Can use underscore (_) in name
  - Can also use characters from other languages, but not recommended
- IPython will output a value of a variable when running a cell with that variable as the last line
  - In regular python, must use a print statement

```
Price
```

A variable named Price

```
"Price"
```

A string literal "Price"

Assignment

- Use assignment statement to "put a value into a variable"
  - In Python, the assignment symbol is =
  - Variable name on the left of =
  - Value on the right of =

```
Price = 500
Value_added_tax = 0.07
```

- Variable must be assigned a value before it can be used
- Variable can be re-assigned to another value

```
Price = 500
Price = 25
```

- In Python, variable has dynamic type. It has the same type as the value assigned to it
  - Can re-assign a new value even if its type is not the same

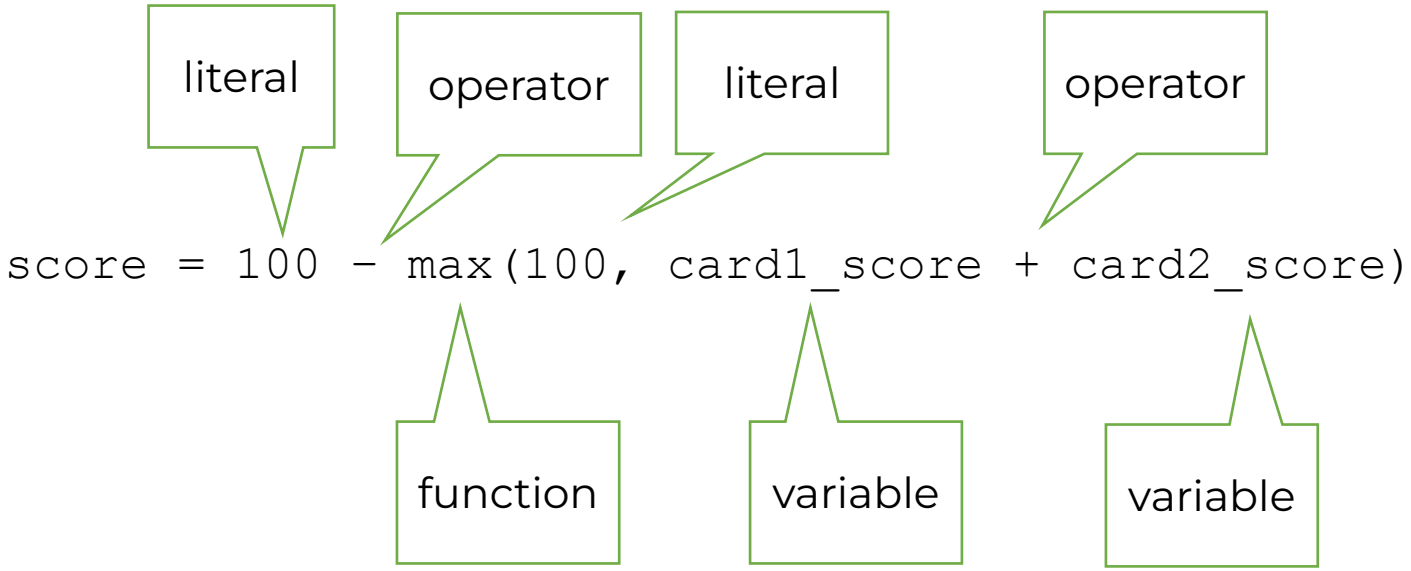
```
Price = 500
Price = " Too Expensive"
```

Python Crash (and incomplete) Course

- Data Type and Variable
- **Expression**
- Function
- Selection and Loop
- Object and method
- List and String
- Dictionary
- Module

Expression

- An evaluation (calculation) that results in a value
- Can be compose from literals, variables, operators, functions, etc.
- Can be used in many places in a program such as on the right side of =



Evaluating an expression

- Calculate values using operators
- Values: things we operate on; operand
  - Literal as itself
  - Variable is referenced to get its value
  - Function is called to calculate its return value (later)
- Operators: what operation we do
  - What operation can be done depends on operands (values)
  - Has operator precedence (order of operation when there are more than one)

Integer and float operators

- Addition and subtraction using + and – symbol
- Multiplication using * symbol
- Exponentiation using ** symbol
- Division: multiple versions
  - True division: / symbol, actual division always results in a float value
  - Floor division: // symbol, `a//b` results in the largest value less than `a/b`
  - Modulo: % symbol, `a%b` results in a remainder of `a/b`
- If any one operand is a float, result is always a float (this is called type coercion from integer to float)
- If all operands are integer, result is integer except the true division

Try it

- `a = 5`
- `f = 4.5`
- `2+a`
- `f-7`
- `f*3`
- `a**2`
- `5/3`
- `5//3`
- `5%3`

Note that the result is float

- These calculation are possible
  - `4**0.5`
  - `2**-1`
- Beware unexpected results
  - `-5//3`
  - `-5%3`
  - `5//-3`
  - `5%-3`

String operators

- String + String: concatenation of two strings
- String * Integer: repeat string equal to the integer
- Try it
  - `"Hello" + " " + "World"`
  - `"Hello"*3`
- Many more string operations later

Comparison operators

- Comparing two expressions resulting in Boolean value (True/False)
- `<` , `<=` , `==` (equal) , `>` , `>=` , `!=` (not equal)
- Try it
  - `5 < 3`
  - `6 >= 4.2`
  - `3 == 3.0` (True due to type coercion)
  - `"Hello" == "hello"` (False: "`H`" and "`h`" are different characters)
- Comparing strings with operators other than `==` and `!=` is not recommended

Confusing == and = is a frequent mistake in programming
- `a == 5` (is a equal to 5?)
- `a = 5` (put 5 in variable a)

Boolean operators

- Operators on Boolean values (`True`/`False`)
- Python has 3 Boolean operators: `and`, `or`, `not`

A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Operator precedence

- Calculate the higher precedence operators first
- If operators have equal precedence, calculate according to associativity
  - `a % b * c equals (a % b) * c`
- Note the only right-to-left associativity of exponentiation
  - `a ** b ** c equals a ** (b ** c)`

Precedence	Associativity	Operator	Description
18	Left-to-right	()	Parentheses (grouping)
17	Left-to-right	*f*(args...)	Function call
16	Left-to-right	*x*[index:index]	Slicing
15	Left-to-right	*x*[index]	Array Subscription
14	Right-to-left	**	Exponentiation
13	Left-to-right	~*x*	Bitwise not
12	Left-to-right	+*x* -*x*	Positive, Negative
11	Left-to-right	* / %	Multiplication Division Modulo
10	Left-to-right	+ -	Addition Subtraction
9	Left-to-right	<< >>	Bitwise left shift Bitwise right shift
8	Left-to-right	&	Bitwise AND
7	Left-to-right	^	Bitwise XOR
6	Left-to-right	|	Bitwise OR
5	Left-to-right	in, not in, is, is not, <, <=, >, >=, <>, == !=	Membership Relational Equality Inequality
4	Left-to-right	not *x*	Boolean NOT
3	Left-to-right	and	Boolean AND
2	Left-to-right	or	Boolean OR
1	Left-to-right	lambda	Lambda expression

Python: Operator of Precedence | Study.com

Missing topics

- Things that will not be taught
  - Bitwise operator
- Things that will be taught later
  - Function and lambda
  - Slicing, indexing (subscription), membership

Python Crash (and incomplete) Course

- Data Type and Variable
- Expression
- **Function**
- Selection and Loop
- Object and method
- List and String
- Dictionary
- Module

Function

- An organized code that can be reused, usually part of a library. Can also be user-defined.
- Creating function using keyword def
- Function body is indented
- Pass parameters and return final value back to caller

Keyword def

Function name

List of parameters

colon

Indented block

Return value

```
[1]  def mySum(i, j, k):
         result = i+j+k
         return result
```

Call function by name

Passed arguments

```
[2]  mySum(5,6,7)
```

Return value

```
18
```

Parameter with default argument

- Define a default value in case no argument passed
- Make the parameter optional
- Must be define after all other parameters

Parameter with default argument

```
[9]  def myCal(i, j, k=3):
         result = i//j+k
         return result
```

No need to pass argument to k

```
[11] myCal(12,5)

     5
```

Argument passing

- Positional arguments: by sequence
- Keyword arguments
  - Can be passed by assigning value to keyword (parameter name)
  - Can be passed out of order
- If using both, must use positional before using keyword

```
[9]  def myCal(i, j, k=3):
         result = i//j+k
         return result
```

```
[11] myCal(12,5)

     5
```

```
[12] myCal(k=2, i=7, j=3)

     4
```

Keyword arguments not in order

```
[13] myCal(10, k=4, j=6)

     5
```

Keyword arguments must come after positional ones

Python Crash (and incomplete) Course

- Data Type and Variable
- Expression
- Function
- **Selection and Loop**
- Object and method
- List and String
- Dictionary
- Module

If statement

- Use keyword if to selectively run or not run code blocks
- Run code in the if block only if the expression (condition) is true

Condition (Boolean expression)

Keyword if

colon

Indented block

Statement outside if

```
[8]  x = 5

     if x > 3:
       x = x+3

     print(x)
```

```
8
```

```
[9]  x = 2

     if x > 3:
       x = x+3

     print(x)
```

```
2
```

if-else

- Choose one of two block
  - Run code blocks under if when condition is true, or alternatively run code blocks under else when condition is false

```
[10] score = 47
     if score > 50:
       print("Pass.")
     else:
       print("Fail.")

     Fail.
```

else keyword at the same level with if. No condition after else

Colon after else

Indented block

if-elif-else

- Choose one from multiple blocks
  - Test conditions of `if` and `elif` one-by-one and run the code block under the first one that is true
  - If no condition is true, run code blocks under `else`

```
[11] score = 65
     if score >= 80:
       print("A")
     elif score >= 70:
       print("B")
     elif score >= 60:
       print("C")
     elif score >= 50:
       print("D")
     else:
       print("Fail.")

     C
```

else keyword at the same level with if. Has condition after else

Indented block

Colon after elif

while

- Run code block under `while` as long as the condition is true
- After finish running the block, check condition again

```
[12] i = 0
     while i < 10:
       print(i)
       i = i + 2
```

```
0
2
4
6
8
```

while keyword with condition

Colon after condition

Indented block is run as long as the condition is true

After the last statement, check condition again

Python Crash (and incomplete) Course

- Data Type and Variable
- Expression
- Function
- Selection and Loop
- **Object and method**
- List and String
- Dictionary
- Module

Class and Object

- Class: a code template for creating object
  - Contains variables and functions that are needed for working
  - Variable inside a class is called attribute
  - Function inside a class is called method
- Object: an actual runtime entity created from a class
  - Usually created by calling class name and passing appropriate arguments
- Can access attribute and method inside an object by using . (dot)

Python Crash (and incomplete) Course

- Data Type and Variable
- Expression
- Function
- Selection and Loop
- Object and method
- **List and String**
- Dictionary
- Module

List

- A data structure consisting of sequence of items
- Sequence has order
- Accessing items by index
- Mutable (can change items in list)
- Can add or remove items from list
- Items can be any type (`integer`, `float`, `string`, etc.)
- Items in list can be of different type

List literals and variables

```
[1] ls = ['a', 'b', 'c', 'd', 'e']
```

```
[3] ls[0]
```

```
'a'
```

```
[4] ls[3]
```

```
'd'
```

```
[5] ls[-1]
```

```
'e'
```

- Created by using [ ] (square bracket)
- Items in list separated by , (comma)
- List literals can be assigned to a variable just like any data
- Accessing an item in a list using its index
  - Index of the first item is 0
  - Negative index is possible, with the last item with index -1
- Put index in [ ] after variable name or list literals
- Index can be an expression that results in integer within the bound of the list

List operations

- Check membership using `in` keyword

```
[6] 'b' in ls
```

```
True
```

- Slicing using index and :, for example ls[x:y:z]
  - x: index to start the slice
  - y: index to stop the slice (item at index y is not included)
  - z: steps

```
[8] ls[1:4:1]
```

```
['b', 'c', 'd']
```

```
ls[0:3:2]
```

```
['a', 'c']
```

```
[11] ls[-1:-3:-1]
```

```
['e', 'd']
```

List and assignment

- Assigning a new value to any item using its index

```
[13] ls[2] = 100
```

```
[14] ls
     ['a', 'b', 100, 'd', 'e']
```

+ operator join 2 lists together, but must assign result to a variable

```
[17] ls = ls + ['k']
```

```
[19] ls.append('l')
```

Use . (dot) after an object to call a method of that object

```
[20] ls.extend(['m','n'])
```

Methods change list in place and does not return anything

```
[21] ls.insert(2, 'z')
```

```
[22] ls
     ['a', 'b', 'z', 100, 'd', 'e', 'k', 'l', 'm', 'n']
```

- Adding new items using
  - + operator
  - append method
  - extend method
  - insert method

List methods

- Removing items using
  - pop method by giving index
  - remove method by giving value of item
- Other methods
  - sort, reverse
  - count, index
- Use help(list) for more info

index of item to pop

```
[23] ls.pop(3)

     100
```

pop removes and returns item at that index

```
[24] ls

     ['a', 'b', 'z', 'd', 'e', 'k', 'l', 'm', 'n']
```

```
[25] ls.remove('z')
```

remove drops the value and does not return anything

```
ls

['a', 'b', 'd', 'e', 'k', 'l', 'm', 'n']
```

List functions

- Functions that can be used with list
  - `len`, `max`, `min`, `sum`
  - Must be able to compare items to use `max`, `min`
  - Must be able to add items together to use `sum`

```
[30] ls = [6,3,5,1,4]

[31] len(ls)

     5

[32] max(ls)

     6

[33] sum(ls)

     19
```

for loop

- Used to iterate over every item in list (and other iterable objects)
- Use an iterator: an object that can be used to traverse a collection from start to end
- No need to set up manual looping using index
- In python, use `for … in` keyword to start iteration

Variable that takes value of item in each iteration

Object that implements iterator protocol (such as list)

for … in keyword

colon

Indented block

```
[34] for x in ls:
         print(x, end=' ')

     6 3 5 1 4
```

List of lists (2d list)

- Items in a list can be another list
- Can be used to simulate 2d-array or a table
- Top-level items can be row or column depending on the library or software used
- Can also create higher-dimension arrays

```
[21] table = [ ['a','b','c'] , ['d','e','f'] ]

[22] table[0][2]

     'c'
```

Tuple

- A data structure consisting of sequence of items
- Sequence has order, accessing items by index
- Immutable (cannot change items in list)
- Items can be any type (integer, float, string, etc.)
- Items in list can be of different type
- Tuple literal is created by , (comma)
  - A parenthesis is not needed, but can be used
- Can use indexing, slicing and iterator like list, but not assignment
- Can also use functions that can be used with list (`len`, `max`, `min`, `sum`)

```
[23] tup = 'a', 7, 6.4

[24] tup

     ('a', 7, 6.4)
```

String

- A data structure consisting of sequence of characters
- Sequence has order
- Accessing items by index
- Immutable (cannot change items in string)
- Cannot add or remove items from string, must create a new string
- Many same characteristics with list
  - Indexing and slicing
  - Iterator

String literals, indexing, slicing, and iteration

```
[45] s = "Hello World"
```

String literal within " " or ' '

```
[46] s[6]

     'W'
```

String indexing using [ ]index starts at 0

```
[47] s[2:9:3]

     'l r'
```

String slicing using [start:stop:step]

```
[49] for c in s:
       print(c.upper(), end=' ')

     H E L L O   W O R L D
```

String iteration using for ... in

upper is a string method changing lower-case character to upper-case one

String methods

- There are tens of methods in string object
- Checking symbol
  - `isupper`, `islower`, `isnumeric`, `isalnum`, `isspace`, etc.
- Changing case
  - `upper`, `lower`, `capitalize`, etc.
- Check/find substrings
  - `startswith`, `endswith`, `find`, `rfind`, etc.
- Manipulation
  - `split`, `strip`, `partition`, `ljust`, `rjust`, etc.
- Use help(str) for more info

Python Crash (and incomplete) Course

- Data Type and Variable
- Expression
- Function
- Selection and Loop
- Object and method
- List and String
- **Dictionary**
- Module

Dictionary

- A data structure consisting of key-value pairs (sometimes called map)
- Has no order
- Accessing items by key
- Value can be any type (integer, float, string, etc.), but key must be immutable (such as integer, float, string, or tuple)
- Values in various pairs can be of different type
- Key must be unique

Dictionary literals and value access

- Use { } to create a dictionary
- Separate key and value by :
- Separate each key-value pair with ,
- Accessing a value by putting key inside [ ]

key

value

item

```
[51] d = {'a':5, 'b':7, 'c':3}

[52] d['b']

     7
```

Access value by key

Listing all keys, all values, and all pairs

- Use methods `keys()`, `values()`, and `items()`
- Return objects are iterable
- Iterate over a dict is the same as iterate over its `keys()`

```
for k in d:
  print(k,":", d[k])
```

```
a : 5
b : 7
c : 3
```

```
[53] d.keys()
```

```
dict_keys(['a', 'b', 'c'])
```

```
[54] d.values()
```

```
dict_values([5, 7, 3])
```

```
[55] d.items()
```

```
dict_items([('a', 5), ('b', 7), ('c', 3)])
```

```
for v in d.values():
  print(v, end=' ')
```

```
5 7 3
```

Python Crash (and incomplete) Course

- Data Type and Variable
- Expression
- Function
- Selection and Loop
- Object and method
- List and String
- Dictionary
- Module

Module

- A group of class templates and variables that can be included in a program
- Can be used to add functionalities to our program
- Can be organized into submodules
- Must install the module and import it into the program/script first
- Use import keyword to import a module
- Caution: module name used during installation and import can be different
- Things not taught: managing modules and environments

Importing modules

- `import`: import whole module
  - Can access class and function inside module using . notation
- `from … import …` : import only one class or function
  - Can use class/function name directly
- `as` keyword used to change name

Module name

Renamed module

```
[70] import pandas as pd
```

```
[71] pd.DataFrame(['a','b'], columns=['x'])
```

	x
0	a
1	b

Accessing DataFrame class inside pandas module

```
[72] from sklearn.neighbors import KNeighborsClassifier
```

submodule

```
[73] knn = KNeighborsClassifier(5)
```

Import only KNeighborsClassifier class

Advanced Topic

- `map`: apply a function to all items in an iterable, return result in an iterable

```
[74] ls

     [6, 3, 5, 1, 4]

[87] import math

[89] list( map(math.sqrt,ls) )

     [2.449489742783178, 1.7320508075688772, 2.23606797749979, 1.0, 2.0]
```

Map returns an iterable, which can be expanded into a list

Apply math.sqrt function to ls.
Note that sqrt has no ()

Lambda

- Used to create a one-use function with one-line expression
- Can be used in places that accept a function (such as map)

```
[96] def sqr(x):
       return x**2

[97] list( map(sqr, ls) )

     [36, 9, 25, 1, 16]

[98] list( map(lambda x: x**2 , ls) )

     [36, 9, 25, 1, 16]
```

lambda keyword

argument

Return value

filter

- Apply function to an iterable and keep only items that returns True

```
[93] ls

     [6, 3, 5, 1, 4]
```

```
[101] list( filter(lambda x: x > 4, ls) )

      [6, 5]
```

- Chaining filter and map

```
[102] list( map(lambda x: x**2,
                filter(lambda x: x > 4, ls)
                )
          )

      [36, 25]
```

`map x**2` to results from filter

filter to `get [6, 5]`

List comprehension

- Combine map and filter in one expression

[ ] to create a list

iteration over ls

```
[103] [x**2 for x in ls if x > 4]

      [36, 25]
```

filter to get [6, 5]

Apply function