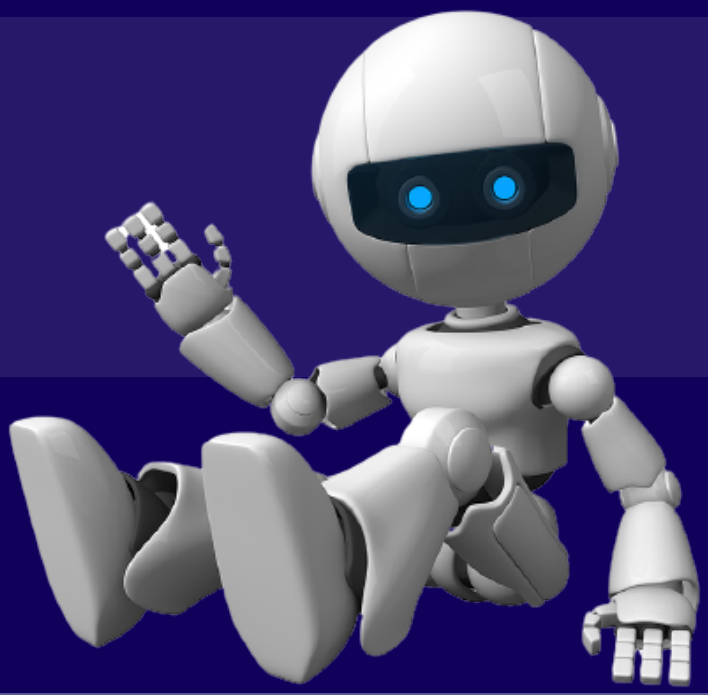
GRSS CHEAT SHEET

THIS GUIDE WILL HELP YOU FAMILIARIZE YOURSELF WITH THE BASICS OF MACHINE LEARNING



MACHINE LEARNING

1- KAGGLE

- Python
https://www.kaggle.com/learn/python
- Learn Pandas Tutorials
https://www.kaggle.com/learn/pandas
- Learn Intro to Machine Learning Tutorials
https://www.kaggle.com/learn/intro-to-machine-learning
- Learn Intermediate Machine Learning Tutorials
https://www.kaggle.com/learn/intermediate-machine-learnin

2- UDACITY

- Intro to Machine Learning
https://classroom.udacity.com/courses/ud120

3-SIMPLELEARN

- Introduction to Machine Learning - A Step by Step Guide
https://www.simplilearn.com/tutorials/machine-learning-tutorial/introduction-to-machine-learning

4- OPENCLASSROOMS

- 'initiez-vous au Machine Learning' (French)
https://openclassrooms.com/en/courses/4011851-initiez-vous-au-machine-learning
- 'Train a Supervised Machine Learning Model'
https://openclassrooms.com/en/courses/6389626-train-a-supervised-machine-learning-model

ADDITIONAL RESSOURCES

- Link to the last session 'Intro to AI' held at SUP'COM
https://supcom-my.sharepoint.com/:v:/g/personal/imtinen_jouili_supcom_tn/Ef7YOcZQWUJOgZFgosnu3egBi9ktgw-oLmb_4F65XcV1sw

REACH US AT:

Email: sbc.supcom.grss@ieee.org
Facebook: IEEE GRSS Chapter-Sup'Com Student Branch

5- COURSERA

- MACHINE LEARNING (Mathematical background)

61-HOUR COURSE + ASSESSMENTS

This course provides a broad introduction to machine learning, datamining, and statistical pattern recognition.

Topics include:

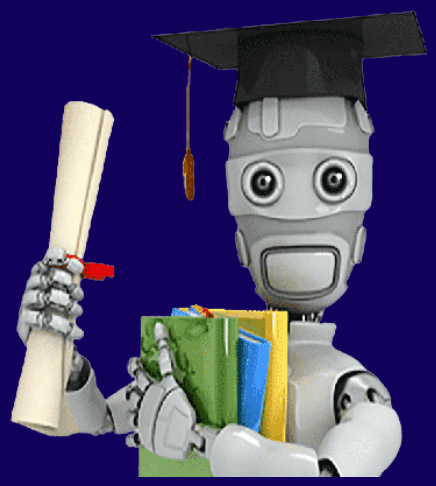
(i) Supervised learning (parametric/non-parametric algorithms, support vector machines, neural network.
(ii) Unsupervised learning (clustering, dimensionality reduction, recommender systems, deep learning).
(iii) Best practices in machine learning (bias/variance theory; innovation process in machine learning and AI).

The course will also draw from numerous case studies and applications, so that you'll also learn how to apply learning algorithms to building smart robots (perception, control), text understanding (web search, anti-spam), computer vision, medical informatics, audio, database mining, and other areas.

https://www.coursera.org/learn/machine-learning

HOW TO GET IT FOR FREE

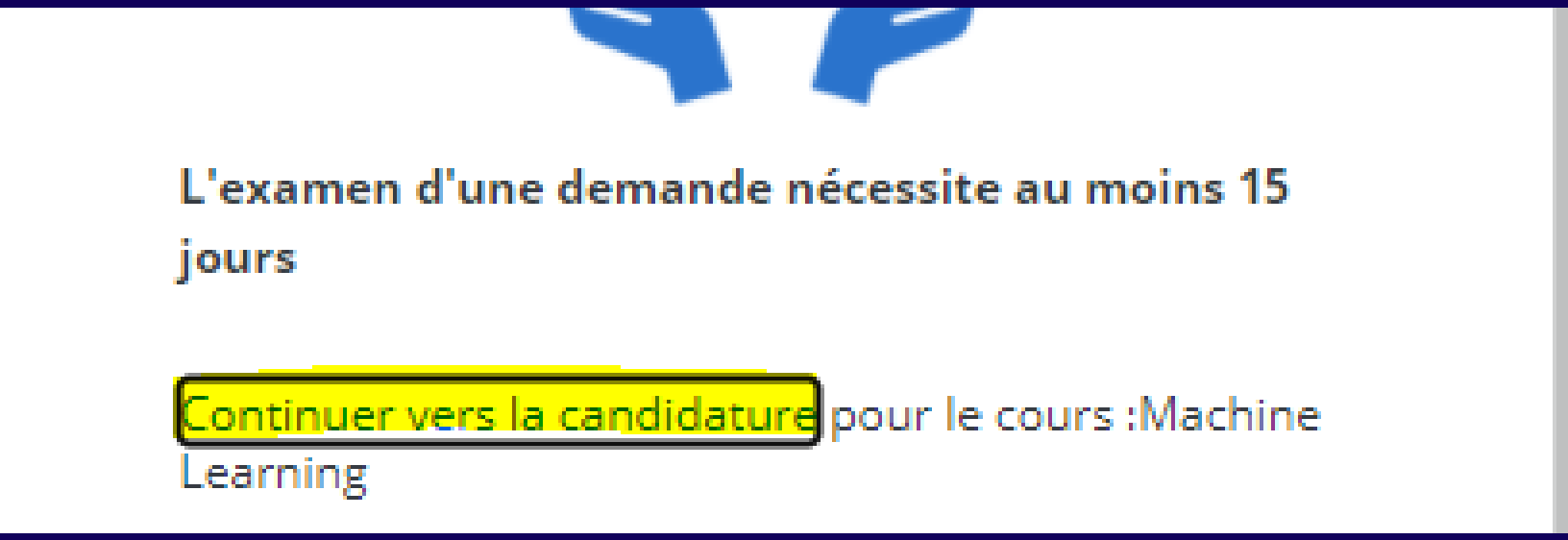
1- Create an account on Coursera
2- Click on the link aforementioned
3- Click 'aide financiaire disponible'



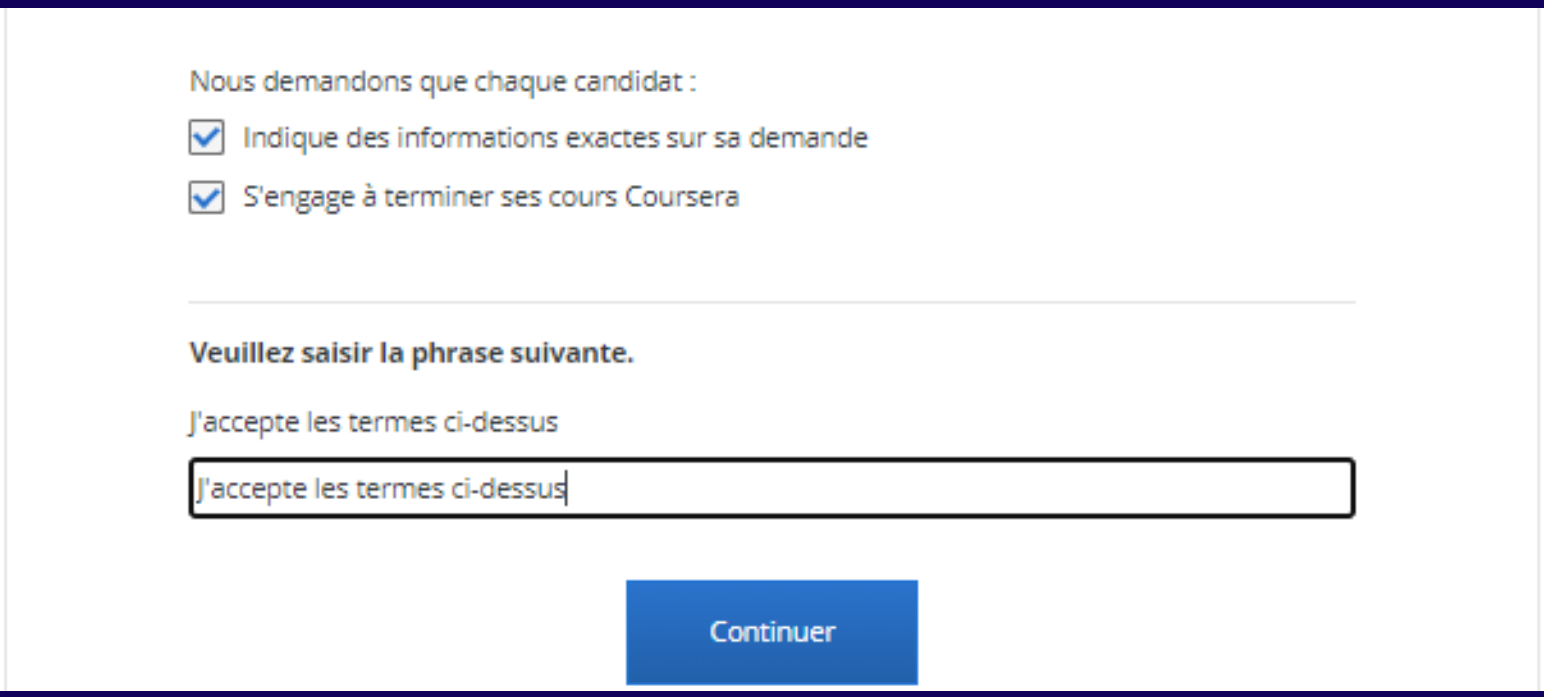
REACH US AT:

Email: sbc.supcom.grss@ieee.org
Facebook: IEEE GRSS Chapter-Sup'Com Student Branch

4- Click on ' continuer vers la candidature'

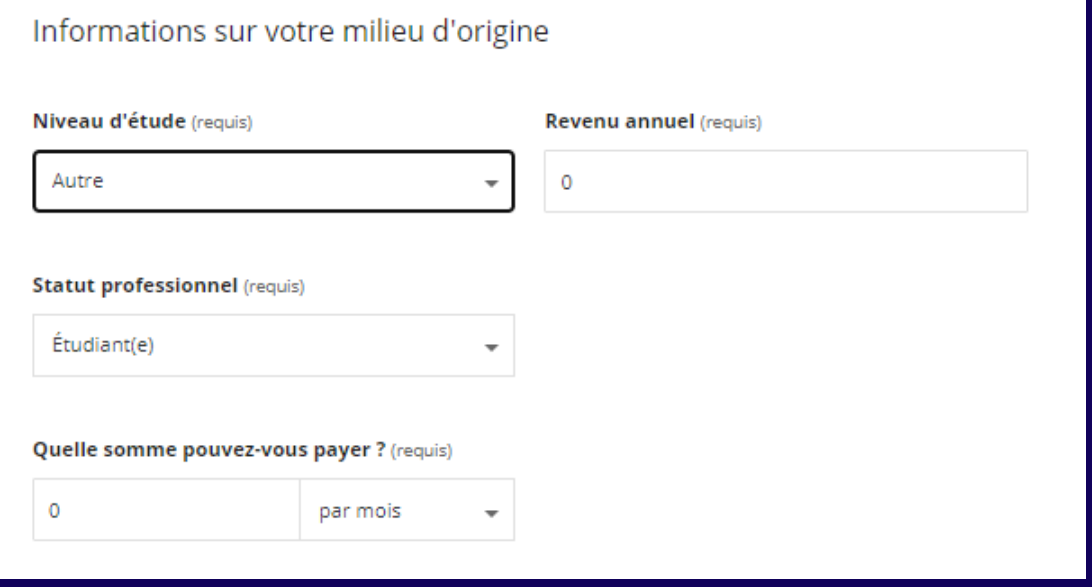


5- Tick both boxes and copy-paste the same sentence then click on the blue button



REACH US AT:

Email: sbc.supcom.grss@ieee.org
Facebook: IEEE GRSS Chapter-Sup'Com Student Branch



6- fill the boxes with the informations shown in the photo

7- Copy and paste the following paragraphs in their three respective cases. It is recommended that you edit the content even a bit in order to guarantee a higher chance of acceptance.

I'm a student from Tunisia and want to learn Machine Learning. I think it will be beneficial for me to get into a good firm as an intern. But I've no job of my own to carry the expanses to pay for the certificate of this course. I live only for my scholarship. In this circumstance, it is very much difficult for me to gather such amount of money for the certificate. Financial Aid will help me take this course without any adverse impact on my monthly essential needs. So I'm badly in need of this financial aid. Receiving this Financial Aid will open for me a new horizons of the world of Coursera courses , which in turn will help me in future .Sir we are three brothers and we all are at learning stage and it becomes a lot difficult for the family members to pay the whole amount for all the three of us and if I add up my course it will make even harder for them to pay. Sir, I need this course very badly for my CV and for increasing my knowledge about the subject

I want to take this course as I want to learn Machine Learning. I want to complete the course due to my curiosity and also that I can put a good CV to a get applied to a job. This Course will boost my job prospects after graduation from my institute. It will help perform better in carrying out various programs in a computer language and give me an edge over my competitors. A verified certificate will attach credibility to the certificate I receive from this course. I plan to complete all assignments on or before time as I have done in previous Signature Track Courses. Also I intend to participate in Discussion Forums, which I have found to supplement my learning immensely in the other online courses I have taken on Coursera. I also plan to grade assignments which are to peer reviewed which I believe will an invaluable learning opportunity.

Sir, the financial status of the family is not too good to pay the amount. We already have a lot of dept in the bank and my parents are paying it on regular basis. It would make their life even harder to add a new money pressure over them. Sir, I don't want to put any pressure over them. Sir, it would be a great help for me to get a good job and help my family if I'm able to get this course.

8- Submit your request and wait for 15 days

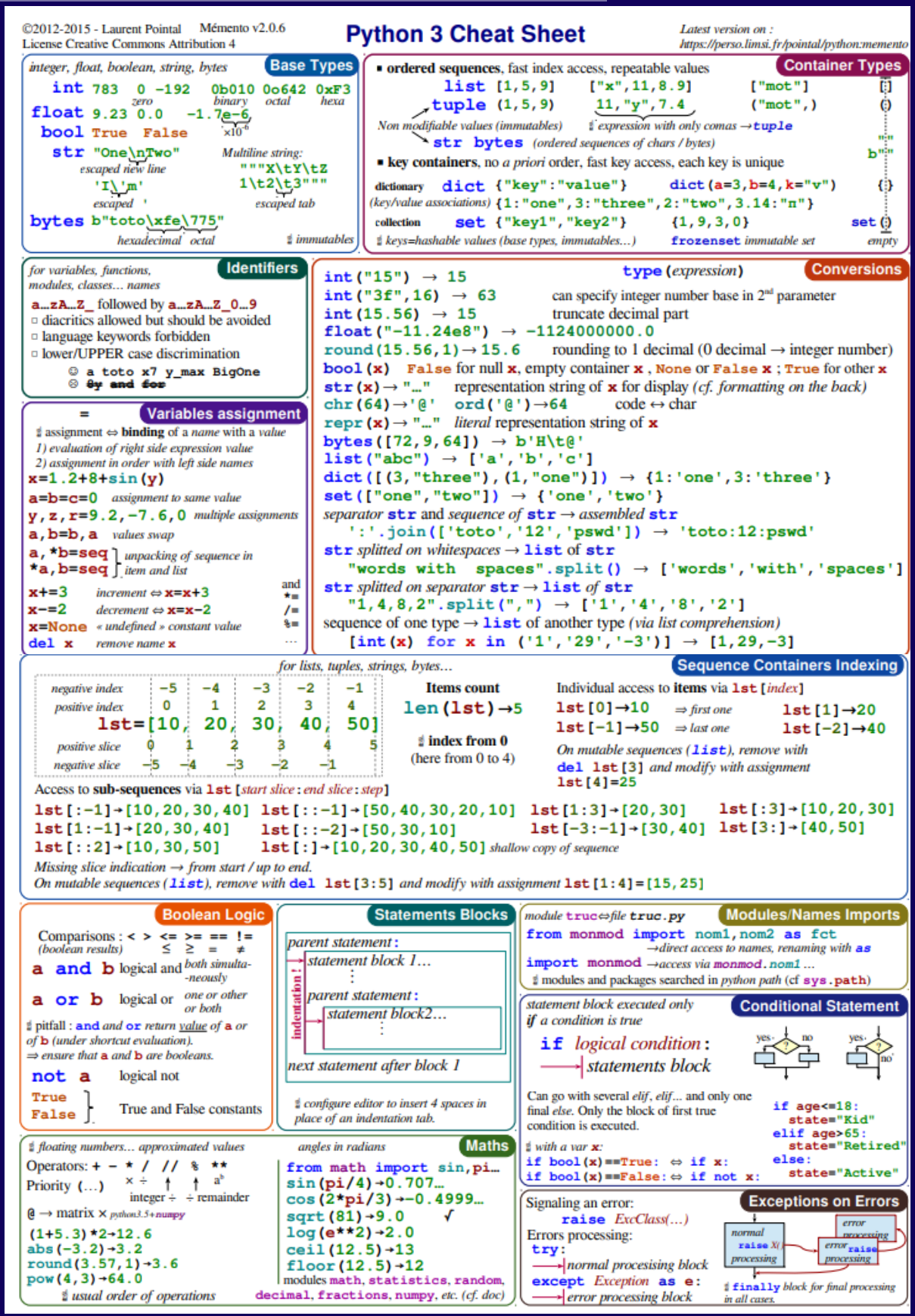
More Advanced Courses Series on Coursera:

- Advanced Machine Learning Specialization: https://www.coursera.org/specializations/aml
- Deep Learning Specialization: https://www.coursera.org/specializations/deep-learning

REACH US AT:

Email: sbc.supcom.grss@ieee.org
Facebook: IEEE GRSS Chapter-Sup'Com Student Branch

6- PYTHON CHEAT SHEET



©2012-2015 - Laurent Pointal Mémento v2.0.6
License Creative Commons Attribution 4

Python 3 Cheat Sheet

Latest version on : https://perso.limsi.fr/pointal/python:memento

Base Types

integer, float, boolean, string, bytes

```
int 783  0 -192  0b010 0o642 0xF3
float 9.23 0.0  -1.7e-6
bool True False
str "One\nTwo"
    'I\'m'
bytes b"toto\xfe\775"
```

zero, binary, octal, hexa; $\times 10^{-6}$; escaped new line; escaped ' ; hexadecimal, octal

Multiline string: `"""X\tY\tZ 1\t2\t3"""` escaped tab

immutables

Container Types

- **ordered sequences**, fast index access, repeatable values

```
list [1,5,9]   ["x",11,8.9]   ["mot"]   []
tuple (1,5,9)  11,"y",7.4     ("mot",)  ()
```

Non modifiable values (immutables); expression with only comas → `tuple`

```
str bytes   ""  b""
```

(ordered sequences of chars / bytes)

- **key containers**, no *a priori* order, fast key access, each key is unique

dictionary (key/value associations)
```
dict {"key":"value"}   dict(a=3,b=4,k="v")   {}
{1:"one",3:"three",2:"two",3.14:"π"}
```

collection
```
set {"key1","key2"}   {1,9,3,0}   set()
```

keys=hashable values (base types, immutables…) `frozenset` immutable set empty

Identifiers

for variables, functions, modules, classes… names

`a…zA…Z_` followed by `a…zA…Z_0…9`
- diacritics allowed but should be avoided
- language keywords forbidden
- lower/UPPER case discrimination

☺ `a toto x7 y_max BigOne`
☹ `by and for`

Variables assignment

`=`

assignment ⇔ **binding** of a *name* with a *value*
1) *evaluation of right side expression value*
2) *assignment in order with left side names*

```
x=1.2+8+sin(y)
a=b=c=0        assignment to same value
y,z,r=9.2,-7.6,0   multiple assignments
a,b=b,a        values swap
a,*b=seq
*a,b=seq       unpacking of sequence in item and list
x+=3           increment ⇔ x=x+3
x-=2           decrement ⇔ x=x-2
x=None         « undefined » constant value
del x          remove name x
```

and `*=` `/=` `%=` …

Conversions

`type(expression)`

```
int("15") → 15
int("3f",16) → 63          can specify integer number base in 2nd parameter
int(15.56) → 15            truncate decimal part
float("-11.24e8") → -1124000000.0
round(15.56,1)→15.6        rounding to 1 decimal (0 decimal → integer number)
bool(x)  False for null x, empty container x , None or False x ; True for other x
str(x)→ "…"   representation string of x for display (cf. formatting on the back)
chr(64)→'@'  ord('@')→64   code ↔ char
repr(x)→ "…"  literal representation string of x
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) → {'one','two'}
separator str and sequence of str → assembled str
  ':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str splitted on whitespaces → list of str
  "words with  spaces".split() → ['words','with','spaces']
str splitted on separator str → list of str
  "1,4,8,2".split(",") → ['1','4','8','2']
sequence of one type → list of another type (via list comprehension)
  [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Sequence Containers Indexing

for lists, tuples, strings, bytes…

negative index	-5	-4	-3	-2	-1
positive index	0	1	2	3	4
lst=[10,	20,	30,	40,	50]	
positive slice	0	1	2	3	4	5
negative slice	-5	-4	-3	-2	-1	

Items count
`len(lst)→5`

index from 0
(here from 0 to 4)

Individual access to **items** via `lst[index]`
```
lst[0]→10   ⇒ first one      lst[1]→20
lst[-1]→50  ⇒ last one       lst[-2]→40
```
On mutable sequences (`list`), remove with `del lst[3]` *and modify with assignment* `lst[4]=25`

Access to **sub-sequences** via `lst[start slice:end slice:step]`
```
lst[:-1]→[10,20,30,40]   lst[::-1]→[50,40,30,20,10]   lst[1:3]→[20,30]   lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]     lst[::-2]→[50,30,10]         lst[-3:-1]→[30,40] lst[3:]→[40,50]
lst[::2]→[10,30,50]      lst[:]→[10,20,30,40,50]  shallow copy of sequence
```
Missing slice indication → from start / up to end.
On mutable sequences (`list`), remove with `del lst[3:5]` *and modify with assignment* `lst[1:4]=[15,25]`

Boolean Logic

Comparisons : `< > <= >= == !=` (boolean results) ≤ ≥ = ≠

`a and b` logical and — both simultaneously
`a or b` logical or — one or other or both

pitfall : `and` and `or` return value of `a` or of `b` (under shortcut evaluation).
⇒ ensure that `a` and `b` are booleans.

`not a` logical not

`True` `False` True and False constants

Statements Blocks

```
parent statement:
    statement block 1…
    :
parent statement:
    statement block2…
    :
next statement after block 1
```

indentation !

configure editor to insert 4 spaces in place of an indentation tab.

Modules/Names Imports

module `truc`⇔*file* `truc.py`

```
from monmod import nom1,nom2 as fct
```
→direct access to names, renaming with `as`
```
import monmod
```
→access via `monmod.nom1` …

modules and packages searched in *python path* (cf `sys.path`)

Conditional Statement

statement block executed only if a condition is true

```
if logical condition:
    statements block
```


Can go with several *elif*, *elif*… and only one final *else*. Only the block of first true condition is executed.

with a var `x`:
```
if bool(x)==True:  ⇔ if x:
if bool(x)==False: ⇔ if not x:
```

```
if age<=18:
  state="Kid"
elif age>65:
  state="Retired"
else:
  state="Active"
```

Maths

floating numbers… approximated values

Operators: `+ - * / // % **`
Priority `(…)` × ÷ ↑ ↑ a^b
integer ÷ ÷ remainder

`@` → matrix × *python3.5+numpy*

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

usual order of operations

angles in radians

```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0   √
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, etc. (cf. doc)

Exceptions on Errors

Signaling an error:
```
raise ExcClass(…)
```

Errors processing:
```
try:
    normal procesising block
except Exception as e:
    error processing block
```


`finally` block for final processing in all cases.

REACH US AT:

Email: sbc.supcom.grss@ieee.org
Facebook: IEEE GRSS Chapter-Sup'Com Student Branch

Conditional Loop Statement

*statements block executed **as long as** condition is true*

```
while logical condition :
    statements block
```

```
s = 0
i = 1
while i <= 100:
    s = s + i**2
    i = i + 1
print("sum:",s)
```

initializations **before** the loop

condition with a least one variable value (here `i`)

make condition variable change !

Algo: $s = \sum_{i=1}^{i=100} i^2$

beware of infinite loops!

Loop Control

`break` immediate exit

`continue` next iteration

`else` block for **normal** loop exit.

Iterative Loop Statement

*statements block executed **for each** item of a container or iterator*

```
for var in sequence :
    statements block
```

Go over sequence's **values**

```
s = "Some text"
cnt = 0
for c in s:
    if c == "e":
        cnt = cnt + 1
print("found",cnt,"'e'")
```

initializations **before** the loop

loop variable, **assignment managed by `for` statement**

Algo: count number of e in the string.

loop on dict/set ⇔ loop on keys sequences

use *slices* to loop on a subset of a sequence

Go over sequence's **index**

- modify item at index
- access items around index (before / after)

```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:",lst,"-lost:",lost)
```

Algo: limit values greater than 15, memorizing of lost values.

good habit : don't modify loop variable

Go simultaneously over sequence's **index** and **values**:

```
for idx,val in enumerate(lst):
```

Display

```
print("v=",3,"cm :",x,",",y+4)
```

items to display : literal values, variables, expressions

`print` options:

- `sep=" "` items separator, default space
- `end="\n"` end of print, default new line
- `file=sys.stdout` print to file, default standard output

Input

```
s = input("Instructions:")
```

`input` always returns a **string**, convert it to required type (cf. boxed *Conversions* on the other side).

Generic Operations on Containers

`len(c)` → items count

`min(c)` `max(c)` `sum(c)`

`sorted(c)` → `list` sorted *copy*

`val in c` → boolean, membership operator `in` (absence `not in`)

`enumerate(c)` → *iterator* on (index, value)

`zip(c1,c2…)` → *iterator* on tuples containing c_1 items at same index

`all(c)` → `True` if **all** `c` items evaluated to true, else `False`

`any(c)` → `True` if **at least one** item of `c` evaluated true, else `False`

*Note: For dictionaries and sets, these operations use **keys**.*

*Specific to **ordered sequences containers** (lists, tuples, strings, bytes…)*

`reversed(c)` → *inversed iterator* `c*5` → duplicate `c+c2` → concatenate

`c.index(val)` → *position* `c.count(val)` → *events count*

```
import copy
copy.copy(c) → shallow copy of container
copy.deepcopy(c) → deep copy of container
```

Integer Sequences

`range([start,] end [,step])`

start default 0, *end* not included in sequence, *step* signed, default 1

`range(5)` → 0 1 2 3 4 `range(2,12,3)` → 2 5 8 11

`range(3,8)` → 3 4 5 6 7 `range(20,5,-5)` → 20 15 10

`range(len(seq))` → *sequence of index of values in seq*

range provides an immutable sequence of int constructed as needed

Operations on Lists

modify original list

`lst.append(val)`	add item at end
`lst.extend(seq)`	add sequence of items at end
`lst.insert(idx, val)`	insert item at index
`lst.remove(val)`	remove first item with value *val*
`lst.pop([idx])` → *value*	remove & return item at index *idx* (default last)
`lst.sort()` `lst.reverse()`	sort / reverse liste *in place*

Function Definition

function name (identifier)

named parameters

```
def fct(x,y,z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

result value of the call, if no computed result to return: `return None`

parameters and all variables of this block exist only *in* the block and *during* the function call (think of a "black box")

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`

**args variable positional arguments (→`tuple`), default values,*

***kwargs variable named arguments (→`dict`)*

Operations on Dictionaries

`d[key]=value`	`d.clear()`
`d[key]` → *value*	`del d[key]`
`d.update(d2)`	update/add associations
`d.keys()` `d.values()` `d.items()`	→ *iterable views on keys/values/associations*
`d.pop(key[,default])` → *value*	
`d.popitem()` → *(key,value)*	
`d.get(key[,default])` → *value*	
`d.setdefault(key[,default])` → *value*	

Operations on Sets

Operators:

- `|` → union (vertical bar char)
- `&` → intersection
- `-` `^` → difference/symmetric diff.
- `<` `<=` `>` `>=` → inclusion relations

Operators also exist as methods.

`s.update(s2)` `s.copy()`

`s.add(key)` `s.remove(key)`

`s.discard(key)` `s.clear()`

`s.pop()`

Function Call

```
r = fct(3,i+2,2*i)
```

storage/use of returned value

one argument per parameter

this is the use of function name *with parentheses* which does the call

Advanced:
**sequence*
***dict*

Operations on Strings

`s.startswith(prefix[,start[,end]])`

`s.endswith(suffix[,start[,end]])` `s.strip([chars])`

`s.count(sub[,start[,end]])` `s.partition(sep)` → *(before,sep,after)*

`s.index(sub[,start[,end]])` `s.find(sub[,start[,end]])`

`s.is…()` *tests on chars categories (ex. `s.isalpha()`)*

`s.upper()` `s.lower()` `s.title()` `s.swapcase()`

`s.casefold()` `s.capitalize()` `s.center([width,fill])`

`s.ljust([width,fill])` `s.rjust([width,fill])` `s.zfill([width])`

`s.encode(encoding)` `s.split([sep])` `s.join(seq)`

Files

storing data on disk, and reading it back

```
f = open("file.txt","w",encoding="utf8")
```

file **variable** for operations

name of file on disk (+path…)

opening **mode**
- `'r'` read
- `'w'` write
- `'a'` append
- …`'+'` `'x'` `'b'` `'t'`

encoding of chars for *text files*: `utf8` `ascii` `latin1` …

cf. modules `os`, `os.path` and `pathlib`

writing

`f.write("coucou")`

`f.writelines(list of lines)`

reading

read empty string if end of file

`f.read([n])` → *next chars* (*if n not specified, read up to end !*)

`f.readlines([n])` → `list` *of next lines*

`f.readline()` → *next line*

*text mode `t` by default (read/write `str`), possible binary mode `b` (read/write `bytes`). **Convert from/to required type** !*

`f.close()` dont forget to **close the file** after use !

`f.flush()` write cache `f.truncate([size])` resize

reading/writing progress sequentially in the file, modifiable with:

`f.tell()` → *position* `f.seek(position[,origin])`

Very common: opening with a guarded block (automatic closing) and reading loop on lines of a text file:

```
with open(…) as f:
    for line in f :
        # processing of line
```

Formatting

formating directives — values to format

```
"modele{} {} {}".format(x,y,r) → str
```

"{*selection* : *formatting* ! *conversion*}"

Selection :

`2`
`nom`
`0.nom`
`4[key]`
`0[2]`

Examples:

```
"{:+2.3f}".format(45.72793)
→'+45.728'
"{2.-ثلاثة}".format(ثلاثة, "ثلاثة")
→'      toto'
"{x!r}".format(x="I'm")
→'"I\'m"'
```

Formatting :

fill char *alignment* *sign* *mini width* . *precision~maxwidth* *type*

`< > ^ =` `+ -` *space* `0` at start for filling with 0

integer: `b` binary, `c` char, `d` decimal (default), `o` octal, `x` or `X` hexa…

float: `e` or `E` exponential, `f` or `F` fixed point, `g` or `G` appropriate (default),

string: `s` … `%` percent

Conversion : `s` (readable text) or `r` (literal representation)

REACH US AT:

Email: sbc.supcom.grss@ieee.org
Facebook: IEEE GRSS Chapter-Sup'Com Student Branch

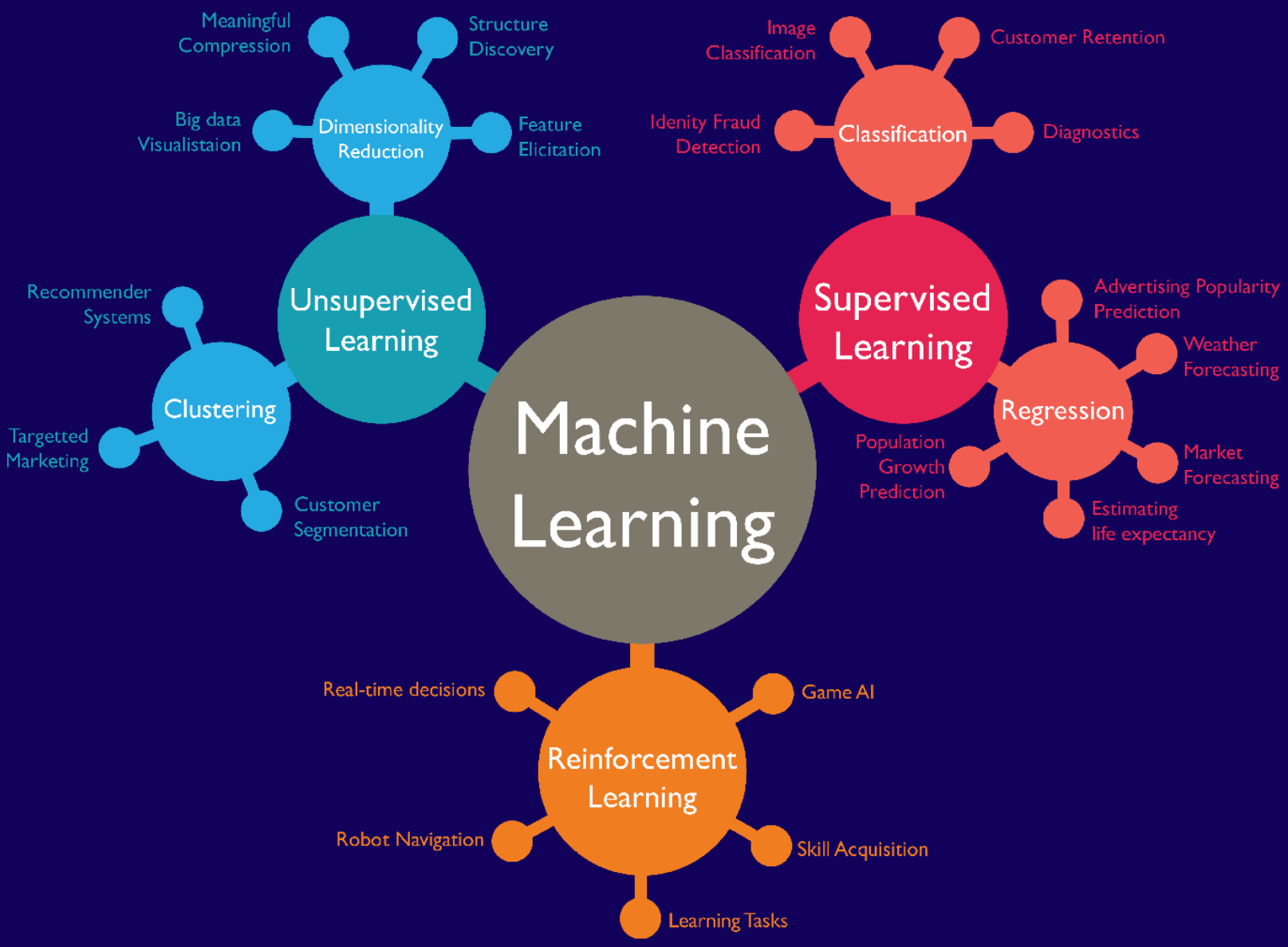
7-MACHINE LEARNING CHEAT SHEET

TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	Linear regression	**The "best fit" line** through all data points. Predictions are numerical.	Easy to understand -- you clearly see what the biggest drivers of the model are.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
Linear	Logistic regression	The adaptation of **linear regression** to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
Tree-based	Decision tree	A graph that uses a **branching method** to match all possible outcomes of a decision.	Easy to understand and implement.	X Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.
Tree-based	Random Forest	Takes the average of many decision trees, each of which is made with a sample of the data. Each tree is weaker than a full decision tree, but **by combining them we get better overall performance.**	A sort of "wisdom of the crowd". Tends to result in very high quality models. Fast to train.	X Can be slow to output predictions relative to other algorithms. X Not easy to understand predictions.
Tree-based	Gradient Boosting	Uses even weaker decision trees, that are increasingly **focused on "hard" examples.**	High-performing.	X A small change in the feature set or training set can create radical changes in the model. X Not easy to understand predictions.
Neural networks	Neural networks	Mimics the behavior of the brain. Neural networks are interconnected neurons that pass messages to each other. Deep learning uses several **layers of neural networks put one after the other.**	Can handle extremely complex tasks - no other algorithm comes close in image recognition.	X Very, very slow to train, because they have so many layers. Require a lot of power. X Almost impossible to understand predictions.

REACH US AT:

Email: sbc.supcom.grss@ieee.org
Facebook: IEEE GRSS Chapter-Sup'Com Student Branch

8-MACHINE LEARNING MINDMAP:



REACH US AT:

Email: sbc.supcom.grss@ieee.org
Facebook: IEEE GRSS Chapter-Sup'Com Student Branch

MACHINE LEARNING IN EMOJI

SUPERVISED UNSUPERVISED REINFORCEMENT

SUPERVISED human builds model based on input / output

UNSUPERVISED human input, machine output
human utilizes if satisfactory

REINFORCEMENT human input, machine output
human reward/punish, cycle continues

BASIC REGRESSION

LINEAR linear_model.LinearRegression()

Lots of numerical data

LOGISTIC linear_model.LogisticRegression()

Target variable is categorical

CLASSIFICATION

NEURAL NET neural_network.MLPClassifier()

Complex relationships. Prone to overfitting
Basically magic.

K-NN neighbors.KNeighborsClassifier()

Group membership based on proximity

DECISION TREE tree.DecisionTreeClassifier()

If/then/else. Non-contiguous data
Can also be regression

RANDOM FOREST ensemble.RandomForestClassifier()

Find best split randomly
Can also be regression

SVM svm.SVC() svm.LinearSVC()

Maximum margin classifier. Fundamental
Data Science algorithm

NAIVE BAYES GaussianNB() MultinomialNB() BernoulliNB()

Updating knowledge step by step with new info

CLUSTER ANALYSIS

K-MEANS cluster.KMeans()

Similar datum into groups
based on centroids

ANOMALY DETECTION covariance.EllipticalEnvelope()

Finding outliers
through grouping

FEATURE REDUCTION

T-DISTRIB STOCHASTIC NEIB EMBEDDING manifold.TSNE()

Visualize high dimensional data. Convert
similarity to joint probabilities

PRINCIPLE COMPONENT ANALYSIS decomposition.PCA()

Distill feature space into components that
describe greatest variance

CANONICAL CORRELATION ANALYSIS decomposition.CCA()

Making sense of cross-correlation
matrices

LINEAR DISCRIMINANT ANALYSIS lda.LDA()

Linear combination of features that
separates classes

OTHER IMPORTANT CONCEPTS

BIAS VARIANCE TRADEOFF

UNDERFITTING / OVERFITTING

INERTIA

ACCURACY FUNCTION $(TP + TN) / (P + N)$

PRECISION FUNCTION $TP / (TP + FP)$

SPECIFICITY FUNCTION $TN / (FP + TN)$

SENSITIVITY FUNCTION $TP / (TP + FN)$

Email: sbc.supcom.grss@ieee.org
Facebook: IEEE GRSS Chapter-Sup'Com Student Branch