

Notes Python

Sources :

- <https://www.tutorialspoint.com/python3/index.htm>

Aller plus loin :

- <https://medium.com/activewizards-machine-learning-company/top-15-python-libraries-for-data-science-in-2017-ab61b4f9b4a7>

Bases

```
#!/usr/bin/python3

# output
print ("Hello, Python!")
print(x, end=" ") # Appends a space instead of a newline in
Python 3

# input
x = input("something:")

# raise exception
raise IOError("file error") #this is the recommended syntax
in Python 3
```

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

Variable Types

```
# Simple Assignment
```

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
value   = 3e+26J       # A complex
name    = "John"       # A string
```

```
# Multiple Assignment
```

```
a = b = c = 1
a, b, c = 1, 2, "john"
```

```
# deletion
```

```
a, b, c = 1, 2, "john"
```

```
str = 'Hello World!'
```

```
print (str)          # Prints complete string
print (str[0])        # Prints first character of the string
print (str[2:5])      # Prints characters starting from 3rd
to 5th
print (str[2:])       # Prints string starting from 3rd
character
print (str * 2)       # Prints string two times
print (str + "TEST") # Prints concatenated string
```

```
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
```

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
```

```
print (tuple)          # Prints complete tuple
print (tuple[0])        # Prints first element of the tuple
print (tuple[1:3])      # Prints elements starting from 2nd
```

```
till 3rd
print (tuple[2:])      # Prints elements starting from 3rd
                        # element
print (tinytuple * 2)  # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple
```

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john', 'code':6734, 'dept': 'sales'}

print (dict['one'])    # Prints value for 'one' key
print (dict[2])        # Prints value for 2 key
print (tinydict)       # Prints complete dictionary
print (tinydict.keys()) # Prints all the keys
print (tinydict.values()) # Prints all the values
```

```
#Data Type Conversion
int(x [,base]) #Converts x to an integer. The base specifies
               #the base if x is a string.
float(x) # Converts x to a floating-point number.
complex(real [,imag]) # Creates a complex number.
str(x) # Converts object x to a string representation.
repr(x) # Converts object x to an expression string.
eval(str) # Evaluates a string and returns an object.
tuple(s) # Converts s to a tuple.
list(s) # Converts s to a list.
set(s) # Converts s to a set.
dict(d) # Creates a dictionary. d must be a sequence of
        #(key,value) tuples.
frozenset(s) # Converts s to a frozen set.
chr(x) # Converts an integer to a character.
unichr(x) # Converts an integer to a Unicode character.
ord(x) # Converts a single character to its integer value.
hex(x) # Converts an integer to a hexadecimal string.
oct(x) # Converts an integer to an octal string.
```

Basic Operators

```

# Python Arithmetic Operators
c = a % b # Modulus
c = a ** b # Exponent
c = a // b # Floor Division

# Python Membership Operators
x in y
x not in y

# Python Identity Operators
x is y
x is not y

```

Decision Making

```

# if statement
if expression:
    statement(s)

# if else statement
if expression:
    statement(s)
else:
    statement(s)

# nested if statements
if expression1:
    statement(s)
elif expression:
    statement(s)
else:
    statement(s)

# Single Statement if
if ( var == 100 ) : print ("Value of expression is 100")

```

Decision Making

```

# while loop
while expression:
    statement(s)

while count < 5:
    count = count + 1
else:
    print (count, " is not less than 5")

# for loop
for iterating_var in sequence:
    statements(s)

>>> range(5)
range(0, 5)
>>> list(range(5))
[0, 1, 2, 3, 4]

fruits = ['banana', 'apple', 'mango']
for index in range(len(fruits)):
    print ('Current fruit :', fruits[index])

for num in numbers:
    if num%2 == 0:
        print ('the list contains an even number')
        break
    else:
        print ('the list doesnot contain even number')

# Loop Control Statements
break
continue
pass

```

```

# Iterators
list = [1,2,3,4]
it = iter(list) # this builds an iterator object
print (next(it)) #prints next available element in iterator

# Iterator object can be traversed using regular for

```

```
statement
for x in it:
    print (x, end=" ")

# or using next() function
while True:
    try:
        print (next(it))
    except StopIteration:
        sys.exit() #you have to import sys module for this
```

Generators

```
import sys
def fibonacci(n): #generator function
    a, b, counter = 0, 1, 0
    while True:
        if (counter > n):
            return
        yield a
        a, b = b, a + b
        counter += 1
f = fibonacci(5) #f is iterator object

while True:
    try:
        print (next(f), end=" ")
    except StopIteration:
        sys.exit()
```

Numbers

```
number = 1 # int
number = 1.2 # float
number = -32.54e100
number = 1 + 2J #complex
number = 0xA0F #Hexa-decimal = 2575
number = 0o37 #Octal = 31
```

Number Type Conversion

```
int(x)
long(x)
float(x)
complex(x)
complex(x, y)

# Mathematical Functions
abs(x)
ceil(x)
cmp(x, y) # Deprecated
exp(x)
fabs(x)
floor(x)
log(x)
log10(x)
max(x1, x2, ...)
min(x1, x2, ...)
modf(x)
pow(x, y)
round(x [,n])
sqrt(x)

# Random Number Functions
choice(seq)
randrange ([start,] stop [,step])
random()
seed([x])
shuffle(lst)
uniform(x, y)

# Trigonometric Functions
acos(x)
asin(x)
atan(x)
atan2(y, x)
cos(x)
hypot(x, y)
sin(x)
tan(x)
degrees(x)
radians(x)
```

```
# Mathematical Constants
pi
e
```

Strings

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

```
# substrings
var1[0]
var2[1:5]
```

```
# String Special Operators
a + b
a * n
a[n] # slice
a[n:m] # range slice
a in b
a in not b
r'...' # raw string
R'...' # raw string
```

```
# String Formatting Operator
print ("My name is %s and weight is %d kg!" % ('Zara', 21))

%c character
%s string conversion via str() prior to formatting
%i signed decimal integer
%d signed decimal integer
%u unsigned decimal integer
%o octal integer
%x hexadecimal integer (lowercase letters)
%X hexadecimal integer (UPPERcase letters)
%e exponential notation (with lowercase 'e')
%E exponential notation (with UPPERcase 'E')
%f floating point real number
%g the shorter of %f and %e
%G the shorter of %f and %E
```

```

*      argument specifies width or precision
-      left justification
+      display the sign
<sp>  leave a blank space before a positive number
#      add the octal leading zero ( '0' ) or hexadecimal
      leading '0x' or '0X',
      depending on whether 'x' or 'X' were used.
0      pad from left with zeros (instead of spaces)
%      '%%' leaves you with a single literal '%'
(var) mapping variable (dictionary arguments)
m.n.  m is the minimum total width and n is the number of
      digits to display
      after the decimal point (if appl.)

```

Triple Quotes

```

para_str = """this is a long string that is made up of
several lines and non-printable characters such as
TAB ( \t ) and they will show up that way when displayed.
the variable assignment will also show up.
"""

```

Unicode String

```

capitalize()
center(width, fillchar)
count(str, beg = 0, end = len(string))
decode(encoding = 'UTF-8', errors = 'strict')
encode(encoding = 'UTF-8', errors = 'strict')
endswith(suffix, beg = 0, end = len(string))
expandtabs(tabsize = 8)
find(str, beg = 0, end = len(string))
index(str, beg = 0, end = len(string))
isalnum()
isalpha()
isdigit()
islower()
isnumeric()
isspace()
istitle()
isupper()
join(seq)
len(string)
ljust(width[, fillchar])

```

```

lower()
lstrip()
maketrans()
max(str)
min(str)
replace(old, new [, max])
rfind(str, beg = 0, end = len(string))
rindex( str, beg = 0, end = len(string))
rjust(width, [, fillchar])
rstrip()
split(str="", num=string.count(str))
splitlines( num=string.count('\n'))
startswith(str, beg=0, end=len(string))
strip([chars])
swapcase()
title()
translate(table, deletechars="")
upper()
zfill (width)
isdecimal()

```

List

```

list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5 ]
list3 = ["a", "b", "c", "d"]

# get
list1[0]
list2[1:5]
L[2]
L[-2]
L[1:]

# set
list[2] = 2001

# remove
del list[2]

```

```

# Basic List Operations
len([1, 2, 3])           # length
[1, 2, 3] + [4, 5, 6]    # concatenation
['Hi!'] * 4              # repetition
3 in [1, 2, 3]           # membership

for x in [1,2,3] : print (x,end = ' ')

# Built-in List Functions and Methods
cmp(list1, list2) # obsolete
len(list)
max(list)
min(list)
list(seq)

list.append(obj)
list.count(obj)
list.extend(seq)
list.index(obj)
list.insert(index, obj)
list.pop(obj = list[-1])
list.remove(obj)
list.reverse()
list.sort([func])

```

Tuples

```

tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5 )
tup3 = "a", "b", "c", "d"

tup1 = (); # empty tuple
tup1 = (50,)

# Accessing Values in Tuples
tup1[0]
tup2[1:5]

# Updating Tuples
tup3 = tup1 + tup2 # concatenate

```

```

del tup

# Basic Tuples Operations
len((1, 2, 3))          # Length
(1, 2, 3) + (4, 5, 6)   # concatenation
('Hi!',) * 4            # repetition
3 in (1, 2, 3)          # membership

for x in (1,2,3) : print (x, end = ' ')

# Indexing, Slicing, and Matrixes
T[2]
T[-2]
T[1:]

# Built-in Tuple Functions
cmp(tuple1, tuple2)
len(tuple)
max(tuple)
min(tuple)
tuple(seq)

```

Dictionary

```

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age']

# Updating Dictionary
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School" # Add new entry

# Delete Dictionary Elements
del dict['Name'] # remove entry with key 'Name'
dict.clear()     # remove all entries in dict
del dict         # delete entire dictionary

# Built-in Dictionary Functions and Methods
cmp(dict1, dict2)
len(dict)
str(dict)

```

```
type(variable)

dict.clear()
dict.copy()
dict.fromkeys()
dict.get(key, default=None)
dict.has_key(key)
dict.items()
dict.keys()
dict.setdefault(key, default = None)
dict.update(dict2)
dict.values()
```

Date and time

```
import time; # This is required to include time module.

ticks = time.time()
print ("Number of ticks since 12:00am, January 1, 1970:",
ticks)

> time.struct_time(tm_year = 2016, tm_mon = 2, tm_mday = 15,
tm_hour = 9,
    tm_min = 29, tm_sec = 2, tm_wday = 0, tm_yday = 46,
tm_isdst = 0)

# Getting current time
localtime = time.localtime(time.time())
localtime = time.asctime( time.localtime(time.time()) ) #
formatted

# The time Module
time.altzone
time.asctime([tupletime])
time.clock( )
time.ctime([secs])
time.gmtime([secs])
time.localtime([secs])
time.mktime(tupletime)
time.sleep(secs)
```

```

time.strftime(fmt[,tupletime])
time.strptime(str,fmt = '%a %b %d %H:%M:%S %Y')
time.time( )

time.timezone
time.tzname

```

```

import calendar

cal = calendar.month(2016, 2)
print ("Here is the calendar:")
print (cal)

# affiche:
Here is the calendar:
    February 2016
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29

# The calendar Module
calendar.calendar(year,w = 2,l = 1,c = 6)
calendar.firstweekday( )
calendar.isleap(year)
calendar.leapdays(y1,y2)
calendar.month(year,month,w = 2,l = 1)
calendar.monthcalendar(year,month)
calendar.monthrange(year,month)
calendar.prcal(year,w = 2,l = 1,c = 6)
calendar.prmonth(year,month,w = 2,l = 1)
calendar.setfirstweekday(weekday)
calendar.timegm(tupletime)
calendar.weekday(year,month,day)

```

Functions

```
# Syntax
```

```

def functionname( parameters ):
    "function_docstring"
    function_suite
    return [expression]

# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return

# Now you can call printme function
printme("This is first call to the user defined function!")
printme("Again second call to the same function")

# All parameters (arguments) in the Python language are
passed by reference.

# Required Arguments
def printme( str ):

# Keyword Arguments
printme( str = "My string")

# Default Arguments
def printinfo( name, age = 35 ):

#Variable-length Arguments
def functionname([formal_args,] *var_args_tuple ):
    "function_docstring"
    function_suite
    return [expression]

# Function definition is here
def printinfo( arg1, *vartuple ):
    "This prints a variable passed arguments"
    print ("Output is: ")
    print (arg1)
    for var in vartuple:
        print (var)
    return

```

```
# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )

#The return Statement
def sum( arg1, arg2 ):
    return total

total = sum( 10, 20 )
```

```
# The Anonymous Functions
lambda [arg1 [,arg2,.....argn]]:expression

# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2

# Now you can call sum as a function
print ("Value of total : ", sum( 10, 20 ))
print ("Value of total : ", sum( 20, 20 ))
```

Modules

```
# The import Statement
import module1[, module2[,... moduleN]]

# support.py
def print_func( par ):
    print "Hello : ", par
    return

#usage
import support

support.print_func("Zara")
```

```
# The from...import Statement
from modname import name1[, name2[, ... nameN]]

# fib.py
```

```
def fib(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result

#usage
from fib import fib
```

```
# The from...import * Statement
from modname import *
```

```
# Executing Modules as Scripts
#!/usr/bin/python3

# Fibonacci numbers module

def fib(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result
if __name__ == "__main__":
    f = fib(100)
    print(f)
```

the following sequences

1. The current directory.
2. If the module is not found, Python then searches each directory in the shell variable PYTHONPATH.
3. If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python3/.

The sys.path variable contains the current directory, PYTHONPATH, and the installation-dependent default.

```
dir(plugin)
locals()
globals()
reload(plugin)
```

```
# Packages in Python
Phone/Pots.py
Phone/Isdn.py
Phone/G3.py

# Phone/__init__.py
from Pots import Pots
from Isdn import Isdn
from G3 import G3

# uses
import Phone

Phone.Pots()
Phone.Isdn()
Phone.G3()
```

Files I/O

```
#Printing to the Screen
print ("Python is really a great language,", "isn't it?")

#The input Function
x = input("something:")
```

```
# The open Function
file object = open(file_name [, access_mode][, buffering])
access_mode = r/r+/w/w+/a, b

# Open a file
fileObject.close();

fo = open("foo.txt", "wb")
print ("Name of the file: ", fo.name)
print ("Closed or not : ", fo.closed)
```

```

print ("Opening mode : ", fo.mode)
fo.close()

# The write() Method
fileObject.write(string);

fo = open("foo.txt", "w")
fo.write( "Python is a great language.\nYeah its great!!\n")
fo.close()

# The read() Method
fileObject.read([count]);

fo = open("foo.txt", "r+")
str = fo.read(10)
print ("Read String is : ", str)
fo.close()

# File Positions
position = fo.tell()
position = fo.seek(0, 0)

```

Objects

```

#!/usr/bin/python

class Employee:
    'Common base class for all employees'
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print "Total Employee %d" % Employee.empCount

```

```

def displayEmployee(self):
    print "Name : ", self.name, ", Salary: ", self.salary

"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount

```

Built-In Class Attributes

```

print "Employee.__doc__:", Employee.__doc__
print "Employee.__name__:", Employee.__name__
print "Employee.__module__:", Employee.__module__
print "Employee.__bases__:", Employee.__bases__
print "Employee.__dict__:", Employee.__dict__

```

Class Inheritance

```

class Parent:          # define parent class
    parentAttr = 100
    def __init__(self):
        print "Calling parent constructor"

    def parentMethod(self):
        print 'Calling parent method'

    def setAttr(self, attr):
        Parent.parentAttr = attr

    def getAttr(self):
        print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class
    def __init__(self):
        print "Calling child constructor"

    def childMethod(self):
        print 'Calling child method'

c = Child()           # instance of child

```

```
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method
```

Overriding Methods

```
class Parent:        # define parent class
    def myMethod(self):
        print 'Calling parent method'

class Child(Parent): # define child class
    def myMethod(self):
        print 'Calling child method'

c = Child()          # instance of child
c.myMethod()         # child calls overridden method
```