A short cheat sheet on Python with the most important commands for IDLE (Integrated Development and Learning Environment)

Installing Packages Using Command Line **Before Using IDLE**

1. Open Command Prompt (Windows: press Win + R, type *cmd*, and press Enter)
2. Type the commands directly:
   py -m pip install numpy
   py -m pip install sympy
   py -m pip install matplotlib

In IDLE start the new file by importing all the functions, classes, and variables from the libraries

```
import numpy as np
import matplotlib.pyplot as plt
```

1.	**print(1+8*3)**	*# displays a message on the screen*
2.	**print(8/2/4)**	*# order of operations 1: (), 2: **, 3: * & /, 4: + & -*
3.	**print(8/(2*4))**	
4.	**print(2*4**2)**	*# exponentiation*
5.	**print(2*pow(4,2))**	
6.	**print(9//2,-9//2)**	*# rounds the result down to the nearest whole number (integer division)*
7.	**print(9%5)**	*# remainder*
8.	**print('abcd\nefgh')**	*# '\n' starts a new line*
9.		
10.	**# VARIABLES & ASSIGNMENT**	
11.	*# **Variable names** must **start with a letter or the underscore***	
12.	*# are **case-sensitive***	
13.	*# can only contain **alpha-numeric characters and underscores***	
14.	**a=4.0**	
15.	**N=int(a)**	
16.	**b=1+a**2**	
17.	**c=abs(a-b)**	*# absolute value*
18.	**eps,x,dt= 1e-10, 100, 0.1**	*# several data in one line*
19.	**print(dt, x, eps, abs(x-123)<eps)**	*# numbers & a logical constant*
20.	**del c**	*# removes the existing variable*
21.		
22.	**s= 1 -1/2 + 1/3 **	
23.	**- 1/4 + 1/5 - 1/6**	*# continuation*
24.	**print(s)**	
25.	**print(round(s,4))**	*# rounds a number to specified number of decimal places*
26.		
27.	**p=input('p=? ')**	*# takes user input as a STRING*
28.	**p=float(p)**	*# conversion to a float number*
29.	**print(p**2)**	
30.	**input('Press <ENTER> to continue')**	
31.		

```
# CONTROL  FLOW
if abs(x-123)<eps:                # <  <=  ==  !=  and or not          IF, ELSE
    print('holds')
else:
    print('does not')

iii = 1
if iii==1:                                                            # IF, ELSE IF (SWITCH)
    print('one')
elif iii==2:
    print('two')
elif iii==3:
    print('three')
else:
    print('????')

sum=0
for i in range(5):                # 0,1,2,3,4  !!!!                   FOR
    print(i)                      # print for each iteration
    sum+=i                        # sum=sum+i
print('suma =',sum)               # prints the final sum

for i in range(2,10,2):    # [2,10)                                   FOR
    print(i,end=' ')       # print with a space for intermediate values

a = 4
while a < 20:                                                         #  WHILE
      print(a)
      a*= 3


#PLOT
plt.ion()                                 # enables interactive mode
x=np.arange(0.01, 2*np.pi, 0.01);
y=np.sin(x)
plt.plot(x, y)                            # plot a graph

x=2.0; y=-0.75; dx=1.0; dy=0.75;
plt.arrow(x, y, dx, dy, width=0.02, color='r')
#plt.xlim(0, 4)                           # plot graph in the selected range
#plt.ylim(-0.8, 1.1)

s = np.linspace(0, 2 *np.pi, 100)
w=np.cos(s)
plt.figure(2)
plt.plot(s,w,'r',linewidth=2,label='Cosine')              # 'r--'
plt.grid(True);  plt.title('My function'); plt.xlabel('X'); plt.ylabel('Y'); plt.legend();
plt.axis('equal')                         # scaling of the axes is the same
#plt.show()                               # displays plot when ion() is not used
```

```
82.  # DEFINE & CALL  A FUNCTION
83.  def f(x):
84.      return np.sin(x)/np.sqrt(x)*np.log(x)
85.
86.  s = np.arange(0.01, 2*np.pi, 0.3);
87.  F=f(s)
88.  plt.plot(s,F, 'mx')                                    # 'mx-'
89.
90.  # A simple lambda function
91.  W = lambda x, y: x + y + 10
92.  print(W(20,15))
93.
94.  # VECTORS AND MATRICES
95.  # Vectors, SINGLE  sq. brackets    [...]
96.  a=np.array([1.0 ,2.0 ,3.0 ])
97.  b=np.array([5. ,4. ,6. ])
98.  print(a,b)
99.  c=a+b
100. print('dot prod = ',a@a,np.dot(a,a))
101. print('cross prod = ', np.cross(a,b))
102. print('squares    ',a*a,a**2)
103. len(a)                          # calculates the number of elements in the vector
104. S=np.sum(a)                     # sum all elements in a
105. print(S)
106. print(min(a),max(a))
107.
108. # Matrices, DOUBLE sq. brackets   [[ ... ]]
109. c=np.array([[1.0, 2.0, 3.0 ]]); d=np.array([[1.0],[2.0],[3.0]])          # row & column matrices
110. print('dot prod. = ',c@d)
111. print('component wise',c*c)
112.
113. A=np.array([[3. , 2. , 0. ],[8.0, 7.0, 6.0 ],[10. , 20. , 30.]])
114. print('matrix A = \n',A)
115. print('A shape:  ',A.shape)                  # dimensions of matrix A
116. print(A.T)                                  # transposition
117.
118. print('A(1,1) ---> A[0,0] = ',A[0, 0])
119. print(np.ones((3, 3)))                      # double parenthesis ((   ))
120. print(np.zeros((1, 3)))
121. print(np.eye(2))                           # identity matrix
122. print(A.max())                             # maximum, max(A)
123. print(A[:,0])                  # prints elements from the first column of matrix A
124. print(A[:,0].reshape(3,1))     # reshape this 1D array into a column vector
125.
126. B = A.copy()                   # Create an independent copy of A
127. A[0,0]=50.0
128. print(A); print(B)
129.
130. v=np.random.uniform(1, 100, size=5)        # vector with 5 random float values in the range [1,100)
131. t=np.random.rand(7)          # an array of 7 random floating-point numbers, each in the range [0, 1)
```

```
# SOLVING A SYSTEM OF LINEAR EQUATIONS
A = np.array([[3. , 1. , 4. ], [1. , 2. , 3. ],[ 2. , 2. , 3.]])
b = np.array([9. , 8. , 7. ])
x = np.linalg.solve(A, b)
print("Solution of the system:\n", x)

# EIGENVALUES
a,b=np.linalg.eig(A)
print('eigenvalues:',a);  print('eigen vectors \n',b)

d=np.linalg.det(A);    print('determinant',d,a[0]*a[1]*a[2])
p=np.linalg.cond(A);   print('condition number=',p)
Ainv=np.linalg.inv(A)                          # inversion of matrix
print('norm A =',np.linalg.norm(A))           # Frobenius norm (Euclidean norm for vector)
print('rank A =',np.linalg.matrix_rank(A))    # the number of linearly independent rows or
                                               # columns in the matrix
```

SYMBOLIC VARIABLES & CALCULATIONS

```
from sympy import *
from sympy.plotting import *
```

```
t=Symbol('t')          # definition of one symbolic variable
x,z= symbols('x z')    # definition of a few symbolic variables
y = x**2+pi*x

Y1=y.subs(x,1.0)       # substitutes the value 1.0 for the variable x in the expression y
Y1f=float(Y1)
print('y(1)= ',Y1,'=',Y1f)

plot(y,(x, 0, 5))                  # command 'plot' is from sympy.plotting library

dy=y.diff(x)                       # first derivative of the function y with respect to the variable x
print('dy',dy)
ddy=dy.diff(x)                     # second derivative
print('ddy',ddy)

I1=integrate(y,x)                  # computes the indefinite integral of the expression y for x
I2=integrate(y,(x,0,1))            #  computes the definite integral of y from 0 to 1
print(I1, I2)

# Define the system of nonlinear equations
eq1 = Eq(x**2 + z**2, 25)          # Circle equation: x^2 + z^2 = 25
eq2 = Eq(x + z, 7)                 # Line equation: x + z = 7
solution = solve((eq1, eq2), (x, z))   # Solve the system of equations
print(solution)

p1=plot(y,(x,0,1),show=False,line_color='blue' ); p2=plot(y,(x,1,2),show=False,line_color='red' )
p1.extend(p2); p1.show()           # Figure with two diagrams p1 and p2
```