**import** module/file (as mo) *(file no extension)*
from module/file import A (as B)
Calling: *module.func or mo.var or A or B*
from module import * *(call original name)*

**help**(function) function?
shift + tab = floating help in Notebook

**pass** = "no-op" statement
**;** =separate commands on same line
**** =escape
**_** = Don't care variable! *(not shown in %who)*

**for** value in sequence:
*for c in "text": for line in open("file"):*
**while** cond :
continue/break
**if** a<b or c>d:
**elif** x>3 & x<5:
**else**:
**with** open(FileName, "r") **as** file_input:

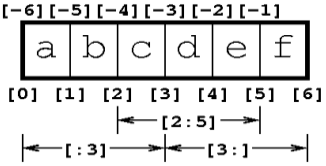
**def** func_name(x,y): *funcs are objects*
*global z x,y are local copy*
*z=z+x*2+y*
*return x*2+y*
**lambda** x,y:x*2+y *anonymous func*
*func_name=lambda x,y: x*2+y*
*name=**lambda** x: y+1 **if** x>0 else 7*
for func in func_array: func_array=[f1,f2]
*func(value)*

**range**([start,] stop[, step])=ret range object
*a=iter(range(10)→a.__next__()→iter can be used 1x*
*a=list(xrange(10))*

**dir**(a)= all visible attributes of object a
*dir() = ret names in current scope (in cmd,out val,var)*
**hasattr**(a, 'split')=does obj a have attr split
**id**(object) = returns object memory address
**len**(object)=length
**type**(a) **isinstance**(a, (int, float))
**str**(a) **bool**(a) **int**(a) **float**(a)

**STRING** **str**(a) **inmutable** char list S[N]=element N
a+b= concatenated string
b=a.**replace**("this", "that")
a.**find**("smth")=lowest index or (-1) if not found
a.**index**("smth")=find with ValueError if not found
a.**capitalize**() a.**lower**() a.**upper**() a.**title**()
a.**count**("aa")=num **nonoverlaping** occurances
a.**isdigit**()=ret bool *'3.45'.isdigit()*
a.**splitlines**()=ret list
a.**split**('^')=ret list def delimiter=" "
'^'.**join**(iterable)=ret concatenated string
a.**rstrip**(['b'])=strip all b at end *(def=whitespace)*
a.**strip**(['b'])=del leading & trailing b

**print** ('a is %s' % (type(a))) ('a is ', type(a))
**template**="%s is no %s"--> template % (a,b)
**%s**=string **%l**=line **%.2f**=float **%03d**=int
""" (or ''') = string over various lines



Negative indices = slice relative to the end
Negative step = reverse direction of moving
*N[-1]=last element N[::-1]=reversed*

**FILE** f=**open**('path' ,['rwa']) (a=append)
*lines=[ ]; for line in f: lines.append(line)*
f.**readlines**()=ret list of lines
f.**writelines**(sequence)=write sequence of strings
f.**read**() = read one line
f.**next**()=next value, or raise StopIteration
f.**write**(str)= Write string to file.
f.**close**() Close the handle
f.**flush**() Flush the internal I/O buffer to disk
f.**seek**(pos)= Move to indicated file position
f.**tell**() =Ret current file position
f.**closed**()=True if the file is closed.

**LIST** L=**list**(a) **L=[ ]=empty** **L[N]**=1 element
*L_copy=L[:]=L*
L.**append**(S) =add 1 element S at the end
L.**extend**([ ]) =append multiple elements
*L=L+[ ]= more expensive (new L created)*
L.**insert**(N,S) =insert element S at position N
L.**remove**(S) =removes first occurrence of S
L.**pop**(N) =del & ret elem N *(default= last elem)*
L.**difference**(L2)=ret elements of L not in L2
S **in** L= ret Bool; *Is element S in list L?*
L.**reverse**() =reverses objects of list in place
L.**sort**(key=method, reverse=T/F)= in-place sort
*(key=str.lower) (key=lambda x:x.count('x'))*

**TUPLE** T=**tuple**(a) **T=( )=empty** **T[N]**
***inmutable,but** content of some elements may change*
*T[1].append('n')→T[1]= same list, but list has changed*
T.**index**('algo') = return first index of 'algo'
T.**count**('S') = num of occurrences of S
**T=T+( )** =concatenate *T+=('ex',)→ok; T+=('ex') →err*
**T=T*N**=concatenate N tuples of T
*T=tuple('string') T=tuple(['string'])*
**(a,b)=(b,a)** → **swap** variable names
T=4,(5,6) **unpack** a,(b,c)=T a,bc=T
*seq=[(1,2,3), (4,5,6)]; for a,b,c, in seq: print a,b,c*

**DICT** **D={key:value}** **D={}=empty→** **D[k]=val**
*KeyValuePair; associative array; K any hashable type*
**D=dict**([(k1,v1), (k2,v2)])=**dict**(**zip**(keys,val))
D.**keys**() = lists keys
D.**values**() = lists values
D.**clear**() = remove all items from dict
**del** D{'key'} = delete from dict
D.**pop**{ k[,d]}=del & return value, [KeyError or d]
D.**popitem**()=del&ret **some** (key, val) KeyErr if {}
D.**items**()=ret list of (key, value) tuples
D.**iteritems**() = iterator over (key, value) items
*iterator=D.iteritems() → iterator.next()*
D.**get**(k, V) =return value of k (V if k not found)
D.**update**(D2)=**merge** D2 to D;**overwrite** if key exists

**SET** **S=set(seq)** **S=set()=empty**
***unordered** set of **unique** elements;(dict **keys only**)*
'k' **in** S= ret Bool, is key in set S
S.**add**(k)=add new element to S
S.**discard**(k)= del k; if not found do nothing.
S.**remove**(k)=del k, if not found KeyError.
S.**update**(seq/S2)= Update as union with seq/S2
S.**issubset**(seq/S2)=is S found in set seq/S2
S.**issuperset**(S2)=is S2 part of S
S.**union**(b)= **S | b**=or
S.**intersection**(b) = **S & b**=and
S.**difference**(b) = **S - b**=
S.**symmetric_difference**(b)= **S ^ b**=nor

**list_comp=[expr for val in collection if condition]**
*res=[x.upper() for x in strings if len(x)>2]*
*res=[]; for x in strings:if len(x)>2:res.append(x.upper())*
**d_comp={k-expr : v-expr for value in coll if cond}**
*key_loc={val:indx for indx, value in enumerate(strings)}*
**s_comp = {expr for value in coll if cond}**
*uniq_length={len(x) for x in strings}*
**Nested comprehensions** *("[x for…" ="return x for…")*
*list_of_tuples=[(1,2,3),(4,5,6),(7,8,9)]*
*flatten=[x for tup in list_of_tuples for x in tup]*
*for tup in list of tuples: for x in tup: return x*

**reversed**(sequence)
**sorted**(sequence[, key=method, reverse=T/F])
**enumerate**(seq[, start])=ret(index, value) of seq
**zip**(sq1 [, sq2 [,..]]) →"pairs" elements to list of tuples
*[(sq1[0], sq2[0] …), (sq1[1],sq2[1]…),…]*
zip(*zipped_seq) =seq1, seq2 …=**unzip**

x ** y = x to the power y = **pow**(x,y)
x**//**y = floor division
x**%**y = reminder of division
**round**(x[, n])= x rounded to n digits, def n=0
math.**floor**(x)= greatest integer as a float <= x
math.**ceil**(x)= least integer as a float >= x

**map**(funct, seq)=apply f to all el of seq→ list
**reduce**(funct,seq[, initial]) =apply f to first 2, then to result & next el till end→scalar
*def mireduce(func, lista):*
*if len(lista) == 0: return None*
*elif len(lista) == 1: return lista[0]*
*else:return func(lista[0], mireduce(func, lista[1:]))*
**filter**(func or None, seq)= filter by boolean function over seq → list,tuple or string

!= **Shell** commands
**a= ! ls** →Slist =Special list type
a.l (or .list)→as list.
a.n (or .nlstr)→as newline-sep string.
a.s (or .spstr)→as space-sep string.
! wc -l **$a.s** =! wc **{a}.s** →python var2shell var
! echo {list(df.columns)} > cols.txt

**%=Magic cmd** %lsmagic %automagic
%cd %quickref %xdel vs del %reset
%who %whos %who_ls
%time list('abcde') vs %%time
%config TerminalInteractiveShell.editor='kwrite'
echo "export EDITOR=kwrite" >>~/.zshrc
%matplotlib inline ->set to work interactively
%precision N= for pretty printing.
%dhist, _dh
%history -n *(num)* -l 100 *(last)* -g *(global)* -f FILE
%edit -x *(do not execute)* %save FILE cells
%rerun -l 10 *(last 10 without this one)*
%macro name 34-36 *(lines)*
23/1-5, ~2/3-4 *(~=session ago)*

**try**:
*expression1*
**except**:
*ErrorName: expression2 (IOError, ValueError…)*

*(sudo) pip install package* *(sudo) pip list*
*conda install package* *conda list*
*sudo pip install -U virtualenv*
*virtualenv dir_name*
*source dir_name /bin/**activate**……**deactivate***
*pip install name(s) →goes in dir_name*
*dir_name/bin/jupyter notebook*
*pandas jupyter keras tensorflow matplotlib seaborn sklearn*