Python for Physicists Quick Reference Guide

Variables

Variable names:
- only contain letters, digits and underscore _, no spaces
- cannot start with a digit
- case sensitive `velocity` different from `Velocity`

Data Types

```
x = 4              # integer
y = 4.2            # float
c = 1 + 2j         # complex
a = 'dog'          # string
happy = True       # boolean

type(x)            # get data type of variable x
```

Conversion between data types:

```
int(x)             # string or float → int
round(2.86)        # rounds float → nearest int
round(2.86,1)      # rounds to tenths place 2.9
float(x)           # int → float
str(3.14)          # int or float → string
```

Print() and Input()

```
print('Hello World')    # prints a message
print(x)                # prints variable x
print('position = ',x) # prints x and message

n = input('Enter n: ') # prompts user for n
```

Print statements display variables and text to command line

```
x = 316.227766          # define a float
print(f'{x:.2f}')       # 316.23
print(f'{x:.4f}')       # 316.2278
print(f'{x:10.2f}')     #     316.23
print(f'{x:.2e}')       # 3.16e+02
```

f-strings pad integers with leading spaces to align:

```
print(f'{x:6d}')        #     31
print(f'{y:6d}')        #   7588
print(f'{z:6d}')        # 102348
```

Print statements can have multiple formatted variables

```
print(f'velocity = {v:.2f}  pos = {x:.2f}')
```

Brackets

```
( )   # parentheses:
      # functions, tuples, grouping
[ ]   # square bracket:
      # lists, arrays, list comprehension
{ }   # curly braces:
      # dictionaries, sets, comprehensions
```

Math using built-in Python

```
z = x + y     # adds x and y
z = x - y     # subtracts y from x
z = x * y     # multiplies x times y
z = x / y     # divides x by y
z = 2**4      # raises 2 to the 4th power

z += 2        # adds 2 to current value of z
z -= 2        # subtracts 2 from z
z *= 2        # multiplies z by 2
z /= 2        # divides z by 2

x // y        # integer division (truncates
              # decimal values)
b = x % y     # remainder of division of x/y
              # also called modulo division
abs(-2)       # absolute value
round(3.14)   # rounds to the nearest integer
```

NumPy math functions

```
import numpy as np # import the numpy library
                   # and abbreviate it as np

# constants
np.pi           # pi
np.e            # e
np.inf          # infinity
np.nan          # not a number

# logarithmic and exponential functions
np.sqrt(x)      # square root(x)
np.exp(x)       # e^x
np.log(x)       # ln(x)
np.log10(x)     # log base 10
np.log2(x)      # log base 2

# trigonometric functions - x in radians
np.sin(x)       # sin(x)
np.cos(x)       # cos(x)
np.tan(x)       # tan(x)

# degree-radian conversions
np.deg2rad(x)   # converts degrees to radians
np.rad2deg(x)   # converts radians to degrees

# inverse trigonometric functions
np.arcsin(x)    # asin(x)
np.arccos(x)    # acos(x)
np.arctan(x)    # atan(x)

# hyperbolic functions
np.sinh(x)      # hyperbolic sin
np.cosh(x)      # hyperbolic cos
np.tanh(x)      # hyperbolic tan
```

Python Lists

- Lists store an ordered collection of items.
- Lists are mutable (they can be modified after created)
- List items don't have to be the same data type

```
A = [10, 20, 30, "hello", True, 60]
```

Python lists index elements starting at 0

```
A[0]      # selects first element (index = 0)
A[1]      # selects 2nd element (index = 1)
A[-1]     # selects last element
A[-2]     # selects 2nd-to-last element
```

Slicing pulls out a subset of a list

```
A[1:4]    # selects elements 1,2,3
A[3:]     # selects elements 3 and after
A[:3]     # selects elements up to but not
          #         including element 3
A[-2:]    # selects last 2 elements of A
A[0:3]=[4,5,7]  # assigns the first 3
                # elements of A to [4,5,7]
```

List methods (functions defined in list objects):

```
A.append(x)   # appends value x to end of A
A.extend(B)   # appends list B to end of A
A + B         # concatenates A and B (does
              # not add elements numerically)
del A[3]      # deletes index = 3 item from A
A.remove(30)  # deletes first element in A
              # whose value = 30
A.insert(n,x) # insert x at index n in list A
A = []        # sets A to the empty list
A.sort()      # sorts items in list A
A.count(x)    # nbr of occurrences of x in A
A[0],A[1] = A[1],A[0]  # swap items 0 and 1
```

Copying lists

```
B = A         # creates an alias to list A,
              # not a copy
B = A.copy()  # creates an independent copy
```

Python functions that work on lists:

```
len(A)        # number of items in list A
sum(A)        # sum of items in list A
min(A)        # minimum value of items in A
max(A)        # maximum value of items in A
```

Python Strings

- Strings are sequences of characters enclosed in quotes
- Strings can be defined with single or double quotes
- Strings are immutable (cannot be modified after created)

```
element = 'carbon'  # define string
element[0]      # first character → "c"
element[-1]     # last character → "n"
element[:3]     # first 3 characters → "car"
len(element)    # Number of characters → 6
```

Because strings cannot be directly modified, we must over-write our old string with a new one to make changes. The following replaces all lower case "c" with upper-case "C":

```
element = element.replace('c','C')
```

```
s = 'cat'       # define string
t = 'hode'      # define another string
s+t             # concatenate strings
                #   → "cathode"
5*t             # repeated copies
                #   → "catcatcatcatcat"
```

```
s = '  A B C  ' # define string
s.lower()       # converts to lower case
s.upper()       # converts to lower case
s.strip()       # removes white space from
                # beginning and end of string
                #   → 'A B C'
s.split()       # parses string into words
                # returned as list elements
                #   → ['A', 'B', 'C']
```

```
s = 'another string example'
s.count('e')    # count the number of e's in
                # string s
```

Dictionaries

Dictionaries replace one variable or string with another

```
# Define your secret code book
codebook = {
    "apple": "meet at midnight",
    "banana": "the mission is on",
    "cherry": "abort the plan",
    "grape": "safehouse secured"
}

print(codebook["apple"])
# prints → "meet at midnight"
```

For loops

- A for loop iterates over a list of items
- body of loop must be uniformly indented

loop over elements of a list:

```
primes = [2, 3, 5, 7, 11]  # define a list
for p in primes:    # loop over items in list
    print(p)        # print value of p
    print(p**2)     # print square of p
```

Use **range()** to loop over sequence of integers

```
for n in range(5):       # loop over integers
    print("n =“,n)       #    0 to 4

for n in range(2,9):     # loop over integers
    print("n =“,n)       #    2 to 9

for n in range(8,2,-1): # loop over integers
    print("n =“,n)       # 8 to 3 (decreasing)
```

Use **enumerate**() to loop over list and list index

- In the following example i is the index and p is the value of the list element

```
primes = [2, 3, 5, 7, 11]
for i,p in enumerate(primes):
    print(f"index = {i}  prime ={p}”)
```

Use **break** to exit a loop early

```
s = 0
for n in range(1000):
   if s+n > 100:      # loop exited if this
        break         # if statement is True
    s += n
print(“sum = “,s)
```

Use **zip**() to loop over multiple lists

```
mass = [0.1, 0.5, 0.9, 2.3]     # masses
vel  = [2.3, 1.2, 3.6, 5.5]     # velocity
KE = []                         # init KE
for m,v in zip(mass, vel):      # loop
    KE.append(0.5*m*v**2)       # find KE
print("KE = ",KE)               # print KE
```

Nested for Loops

Example: print multiplication table

```
N = 10                          # N = table max
for row in range(1,N+1):        # loop over rows
    for col in range(1,N+1): # loop over cols
       # print product, suppress new line
       print(f”{row*col:3d}  “,end=“”)
    print()                     # print new line
```

while loop

- Keeps looping while some condition remains true.

Example: Print out all powers of 2 less than 100

```
p = 1               # initialize power
while p < 100:      # loop while power is less
                    # than 100
    print(p)        # display current value
    p = p * 2       # multiply power by 2
```

list comprehension

```
squares = [x**2 for x in range(10)]
```

if statement

- An if statement controls whether a block of code is executed
- The first line starts with if and ends with a colon
- The body containing one or more statements is indented
- If statements can stand alone or they can be combined with else if (`elif`) and `else` statements.

```
if a > b:
    print("a is greater than b”)
```

In this example, we combine `if`, `elif` and `else` statements:

```
if a > b:
    print("a is greater than b”)
elif a < b:
    print("b is greater than a”)
else:
    print("a must be equal to b”)
```

Logic

Comparison operators return True or False:

==	equal	!=	not equal
<	less than	<=	less than or equal equal
>	greater than	>=	greater than or equal equal
is	same object	is not	not same object
in	membership	not in	not a member

Boolean operators (combine conditions):

`and` `or` `not`

Examples:

```
5 >= 0              # True
5 > 6               # False
3 > 2 and 5 > 4     # True
5 in [4,5,6]        # True
‘cat’ in ‘cathode’  # True
‘dog’ in ‘cathode’  # False
```

Coding Patterns with Loops

- Coding patterns are commonly used combinations of loops, if statements, variable updates, etc. to achieve particular goals.

Accumulator Pattern

- The Accumulator Pattern sums or combines elements in a loop.

Example: sum numbers 1 to 10:

- `N` sets the number we count to
- `tot` will be a running total as we add up the numbers
- Because we keep adding to `tot`, this is an Accumulator Pattern

```
N = 10                    # number to sum to
tot = 0                   # accumulator
for i in range(1,N+1):    # loop i = 1→10
    tot = tot + i         # running total
print("sum 1 to”,N,"is",total)  # show result
```

Example: create a sequence of numbers starting at $p_0 = 1$ such that each element is double the previous element.

- p is a list that will contain our sequence
- We initialized p to have a single element = 1
- In each iteration of the loop, we add one more element onto the list p, whose value = previous element * 2
- This is an Accumulator Pattern because the list p keeps accumulating new elements.

```
p = [1]                   # initialize the list
for n in range(10):       # loop n = 0→9
    p.append(p[-1]*2)     # next item =
                          #   previous item * 2
print(p)                  # display list
```

Update Pattern

- The Update (or Replacement) Pattern updates old information with new

Example: Finding the max or min of a list

- the variable `max_value` will store the maximum value in the list
- We initialize it to the first element
- In each iteration of the loop, we check to see if the list element is larger than `max_value`
- If it is larger, then we update `max_value` with that larger value.

```
A = [3, 7, 9, -5, 27, -2, 12]
max_value = A[0]              # initialize
for num in A:                 # loop over A list
    if num > max_value:       # check if value
        max_value = num       # is > max_value
print("maximum value = ", max_value)
```

Example: Finding the difference between pairs of items

- The `y_old` and `y_new` variables are updated in each iteration of the loop

```
y_values = [0, 1, 5, 4, 2, 0]  # define list
y_old = y_values[0]            # initialize
y_new = y_values[0]
for y in y_values:             # loop
    y_old = y_new              # update
    y_new = y
    diff = y_new - y_old
    print('diff = ',diff)
```

Count Pattern

- Use a counter variable to count the number of occurrences

Example: Given a list of the numbers of cats owned by different people, count how many folks own at least 3 cats

- `count` is a variable that will be incremented every time someone owns at least 3 cats
- The if statement inside the loop checks every item in the list to see if it is 3 or more. If yes, `count` is incremented

```
n_cats = [0,1,5,0,1,10]     # number of cats
count = 0                   # initialize count
for cats in n_cats:         # loop over list
    if cats >= 3:           # check condition
        count = count + 1   # increment count
print(count," people own at least 3 cats”)
```

Search Pattern

- Use a counter variable to count the number of occurrences

Example: Given a list of the numbers of cats owned by different people, count how many folks own at least 3 cats

- The `break` command exits the loop if the target is found.

```
numbers = [3,11,15,24]    # list of numbers
target = 15               # search for this
found = False             # init found
for n in numbers:         # loop over list
    if n == target:       # if target is found
        found = True      # set found to True
        break             # exit the loop
if found:
    print(target,”was found”)
else:
    print(target,”was not found”)
```