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.asin(x)      # arcsin(x) also works
np.acos(x)      # arccos(x)
np.atan(x)      # arctan(x)
np.atan2(y,x)   # arctan given x,y components

# 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 8

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”)
```

Loop Examples (version 2)

for Loop - Iterate over a list or NumPy array

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

for Loop - 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 8

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

for Loop - Use enumerate() to loop over list & index

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

for Loop - Use break to exit 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)
```

for Loop - Use zip() to loop over multiples 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 Loop

```
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

```
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
```

The Accumulator Pattern: Sum integers

```
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
```

The Accumulator Pattern: Sum integers

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

Count Pattern: Count # occurrences in a list

```
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”)
```

Update Pattern: Find the maximum of an array

```
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)
```

Update Pattern: Find the maximum of an array

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

Search Pattern: Look for a given value in a list

```
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”)
```

NumPy Arrays - 1D

Creating a 1D NumPy Array from a List

```
A = np.array([10, 20, 30, 40, 50])
```

Copying NumPy Arrays

```
B = A                    # B is an alias of A
B = A.copy()             # B is a copy of A
```

Fetching Array Contents and Slicing

```
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 an Array

```
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]
```

Vectorized Operations (element-by-element)

Most NumPy functions are Vectorized. A few examples:

```
x*y              # element-by-element product
x**2             # element-by-element square
np.sqrt(x)       # square root(x)
np.exp(x)        # e^x
np.sin(x)        # sin(x)
np.atan2(y,x)    # arctan given x,y components
```

NumPy Array Attributes

```
A.size           # number of elements
A.ndim           # number of dimensions
A.shape          # tuple, length of each axis
A.dtype          # data type of the array
```

NumPy Array Statistics Methods

```
A.sum()          # sum of the elements
A.prod()         # product of the elements
A.mean()         # mean of the elements
A.std()          # standard deviation
A.var()          # variance
A.min()          # minimum value
A.max()          # maximum value
A.cumsum()       # cumulative sum
A.cumprod()      # cumulative product
```

Create Linearly-space NumPy Arrays

```
# specify number of points
np.linspace(start, stop, N)
```

```
# specify interval between points
np.arange(start, stop, interval)
```

Create NumPy Array filled with 0's or 1's

```
np.zeros(N)      # 1D array with N elements
np.ones(N)       # 1D array with N elements
```

Create NumPyArray filled with random numbers

```
# create random number Generator
rng = np.random.default_rng()

# Random Integers
rng.integers(low, high, size=N)
rng.integers(low, high, size=N,endpoint=True)

# draw float from uniform or normal distrib.
rng.normal(size=N)
rng.uniform(low, high, size=N)

# randomly choose from a list of items
rng.choice(my_list, size=N)
rng.choice(10,size=5,replace=False)
```

Logical Indexing

```
A[A>50]      # elements of A with A > 50
B[A>50]      # elements of B with A > 50
mask = A>50  # define a mask to use
A[mask]      # elements of B with A > 50
A[(A>10) & (A<50)]  # combine with AND (&)
A[(A<10) | (A>50)]  # combine with OR (|)
```

Logical Assignment

```
A[A<0] = 0                  # A<0 set to A=0
B = np.where(A>0, 1, -1)    # assign values to
                            # B based on A
```

NumPy Arrays - 2D

Fetching and Slicing 2D Arrays

```
A[1,2]    # element with row=1, col=2
A[0,:]    # selects top row
A[:,-2:]  # selects last two columns
```

Create 2D NumPy Arrays

Tuple specifies # rows (n) and columns (m)

```
np.zeros((n,m))  # nxm array filled with 0's
np.ones((n,m))   # nxm array filled with 1's
np.ones(A.shape) # same shape as array A
```

Working with 2D Arrays

```
A.T              # transpose
A.flatten()      # convert to 1D array
A.reshape((n,m)) # change shape of an array
A.mean(axis=0)   # calc. mean along columns
A.std(axis=1)    # calc. std along rows
```

`x = [1 2 3]`	set `x` to be a 1x3 row vector
`x = [1, 2, 3]`	same as above (1x3 row vector)
`x = [1; 2; 3]`	set `x` to be a 3x1 column vector
`x = [1 2; 3 4]`	set `x` to be a 2x2 matrix
`x = []`	removes contents of variable `x`

Create a Vector Using Colon Notation

`1:5`	creates row matrix [1 2 3 4 5]
`0:2:10`	count by twos [0 2 4 6 8 10]
`5:-1:0`	count backwards [5 4 3 2 1 0]
`0:pi/10:pi/2`	increments of pi/10 from 0 to pi/2

More Ways to Create Matrices

`linspace(a,b,n)`	n equally spaced numbers from a to b
`logspace(a,b,n)`	n log-spaced numbers from 10^a to 10^b
`zeros(2)`	fill a 2x2 matrix with zeros
`zeros(1,3)`	fill a 1x3 row vector with zeros
`ones(2,3)`	fill a 2x3 matrix with ones
`eye(3)`	3x3 identity matrix I
`rand(1,10)`	fill a 1x10 row vector with uniform random numbers on [0,1)
`randi(N,1,10)`	fill a 1x10 row vector with random integers between 1 and N

Referencing Cells in a Row or Column Vector

`x(3)`	3rd element of vector `x`
`x(3:8)`	3rd to 8th elements of `x`
`x(end)`	last element of `x`
`x(3:end)`	3rd to last element of `x`
`x(1:2:end)`	odd elements of x, i.e. 1, 3, 5,...
`x(end:-1:1)`	returns elements in reverse order

Referencing Cells in an n x m Matrix

`A(2,3)`	cell in 2nd row, 3rd column of matrix `A`
`A(5,:)`	5th row of x → row vector
`A(:,4)`	4th column of x → column vec.
`A(1:2,3:4)`	extracts a 2x2 matrix
`A(:,[1 3])`	1st and 3rd columns of matrix `A`

Operations with Vectors and Matrices

`x * 3`	multiply every element of `x` by 3
`x + 3`	add 3 to every element of `x`
`x .* y`	element-wise product of vectors `x` and `y` (`x` and `y` must be same length)
`x * y`	inner (dot) product of row vector `x` and column vector `y`
`A * y`	matrix product of matrix `A` and vec `y`
`x.^2`	square very element of `x`
`sqrt(x)`	square root of every element of x
`sin(x) ./ x`	element-wise calculation of $\sin(x)/x$

Magnitude of a Vector

`norm(x)`	norm (magnitude) of a vector
`vecnorm(x)`	norm (magnitude) of a group of vectors stored as columns in matrix x
`vecnorm(x')`	norm (magnitude) of a group of vectors stored as rows in matrix x

Matrix Properties and Stats

`size(A)`	# rows and # columns of matrix `A`
`length(x)`	# elements in a vector or maximum dimension of a matrix
`sum(x)`	sum of elements in vector x
`min(x)`	minimum value of cells in vector x
`max(x)`	maximum value of cells in vector x
`[xmin,imin]=min(x)`	min value and index of vector x
`[xmax,imax]=max(x)`	max value and index of vector x
`mean(x)`	mean value of cells in vector x
`std(x)`	standard deviation of vector x
`mode(x)`	mode (most common value) of vector x

Transpose and Adjoint Operators

`x.'`	transpose of vector `x` (converts column vector ↔ row vector)
`x'`	adjoint of vector x (complex conjugate of transpose)

Dot Product and Outer Product

Let `x` and `y` be $1 \times n$ row vectors

`x * y'`	inner (dot) product of two row vectors `x` and `y`
`x' * y`	outer product of a column vector `x'` and a row vector `y`
`dot(x,y)`	another way of writing the dot product works even if `x` and `y` are both row or column vectors

Matrix Multiplication

Let `x` and `y` be two matrices

`x * y`	matrix multiplication of x and y

Reshaping Matrices

Let `x` be an $n \times m$ matrix

`reshape(x,p,q)`	reshape x as a $p \times q$ matrix. Default is to read down consecutive columns. Product $n \cdot m$ must equal product $p \cdot q$
`reshape(x,1,[])`	reshape x as a row matrix: first n elements are the first column of x, next n elements are second column, etc.
`reshape(x',1,[])`	reshape x as a row matrix: first m elements are the first row of x, next m elements are second row, etc.
`repmat(x,p,q)`	create a matrix filled with $a \times b$ copies of matrix x: p copies in first dimension, q copies in second dimension

Writing Matlab Scripts

Functions

- Functions let you execute a block of code using a single call statement.

example: prints "Hello!"

```
def print_greeting():    # define function
    print('Hello!')      # body of function

print_greeting()         # call the function
```

example with passed parameters:

```
def print_date(year, month, day):
    print(f'{month}/{day}/{year}')

# pass the parameters in order
print_date(1871, 3, 19)

# specify parameters by name
print_date(day=3, month=19, year=1871)
```

example with returned value:

```
def average(values):
    return sum(values) / len(values)

ave = average([1, 3, 4])
    print('ave = ',ave)
```

Libraries

Built-in Python may be extended with many libraries, including:

- numpy - numerical calculations
- matplotlib - scientific plotting
- seaborn - extends matplotlib
- random - random numbers
- pandas - handling large datasets:
- scipy - scientific computation

Script File Names

- Script name cannot start with a number:
  - `33abc.m` Not Allowed
  - `abc33.m` Allowed
- Script names cannot contain spaces or periods, but hyphens or underscores are OK:
  - `abc 33.m` Not Allowed
  - `abc.33.m` Not Allowed
  - `abc_33.m` Allowed
  - `abc-33.m` Allowed
- Do not use names of existing Matlab functions:
  - `sin.m` Not Allowed
  - `sin_func.m` Allowed

Housekeeping

Place these commands at the beginning of your scripts:

`clear`	clears all variables from memory
`close all`	close all figure windows
`clc`	clear command window
`figure`	create a new figure window

Comments

The "`%`" sign may be used to document your code

```
x0 = 3;            % initial value of x
```

Wrap Long Lines

Use three periods to continue a long ling onto the next line.

```
x = a0 + a1 * h + (1/2) * h^2 + ...
   (1/6) * h^3;
```

Input

Prints a message to the command line asking the user to enter a number.

```
N = input('enter N:  ');
```

Saving Workspace Variables

To save all variables in your workspace either click "Save Workspace" under the "Home" tab. Or use the `save()` command. The saved file will have extension `.mat`.

```
save('my_variables');
```

To save just two variables `x` and `y` use (note the variable names must be in single quotes):

```
save('my_variables', 'x', 'y');
```

Loading Saved Workspace Variables

To load saved variables back into memory simply double-click on the `.mat` file within Matlab. You may also use the load() command:

```
load('my_variables.mat');
```

Display Text or Variable

`disp("hello world")`	Prints "hello world" to the command window
`disp(x)`	Displays the value of x
`disp(x,FORMAT)`	Displays the value of x with formatting (see below)

Formatted Output

`fprintf(FORMAT, variables)`

Prints the variables to the command line based on formatting specified in the `FORMAT` string:

`%i`	integer	`%e`	scientific (exponential)
`%f`	floating point	`%s`	string
`%g`	tries to use most concise, easy-to-read format		

Loops and If Statements

Special characters:

`\t` horizontal tab `\n` new line

```
str = sprintf(FORMAT, variables)
```

Same as `fprintf()` but it saves the result to a string variable called `str` instead of displaying the result to the screen.

Formatted Output Examples

```
a = 3
fprintf('a = %i\n', a)      prints:  a = 3
fprintf('a = %4i\n', a)     prints:  a =    3
fprintf('a = %04i\n', a)    prints:  a = 0003

b = sqrt(2)
fprintf('b = %g\n', b)      prints:  b = 1.41421
fprintf('b = %.3g\n', b)    prints:  b = 1.41
fprintf('b = %.3g\n', b)    prints:  b = 1.4142136

fprintf('b = %.3f\n', b)    prints:  b = 1.414
fprintf('b = %.3e\n', b)    prints:  b = 1.414e+00
```

Common Errors

"inner matrix dimensions must agree"

You multiplied two matrices improperly. If you wanted to multiply them element-by-element, make sure you used **".*"** rather than **"*"**. Make sure the matrices have the same length.

"index exceeds matrix dimensions"

You tried to access an element of an array that was larger than the array, i.e. `x(3)` gives the error if `x = [1 2]`

"undefined function or variable"

If you get this message, check your spelling and case. Most Matlab commands are lower case.

ctrl-C

Stops runaway code

Sometimes a calculation produces one of the following:

- `inf` = infinity (produced if you type `1/0`)
- `NaN` = Not a Number (produced if you type `0/0`)
  in this case the result is undefined

`for` Loops

Sum numbers 1 through 10

```
x = 0;
for i=1:10
  x = x + i;
end
```

Nested `for` Loops

Example: create multiplication table and store in matrix A

```
N = 13;
for i=1:N
   for j=1:N
      A(i,j) = i*j;
   end
end
```

Logic

Logical elements:

`==`	equal	`~=`	not equal
`<`	less than	`<=`	less than or equal equal
`>`	greater than	`>=`	greater than or equal equal
`&`	and	`|`	or
`~`	not		

`if` statement

The statements between the if statement and the end statement are executed only if the condition is true.

example:

```
if a > b
  disp('a is greater than b')
end
```

The `else` statement is executed if the condition is false.

example:

```
if a > b
  disp('a is greater than b')
else
  disp('a is less than or equal to b')
end
```

The `else if` allows for additional conditions.

example:

```
if a > b
  disp('a is greater than b')
else if a < b
  disp('a is less than b')
else
  disp('a must be equal to b')
end
```

`while` loop

Keeps looping while some condition remains true.

This example finds the smallest power of 2 greater than 1000.

```
x = 1;
while x <= 1000
  x = x * 2;
end
disp('smallest power of 2 > 1000 is')
disp(x)
```

Measuring Elapsed Time

Place code you want to clock between the commands `tic` and `toc`. The following example measures the time to generate 100 million random numbers.

```
tic
x = rand(1,100000000);
toc
```

Runaway Code

If your program gets stuck in a loop, click in the control window and hit Ctrl-C

Anonymous Functions

This example function in named "`multiply`" . It takes the variables `a` and `b` as input and returns their product.

```
multiply = @(a,b) a .* b;
```

To use this function to multiply 3×4 do the following:

```
multiply(3,4)
```

m-file Functions

This example shows how to write the Anonymous function `multiply` as an m-file function.

```
function y = multiply(a,b)
       y = a .* b;
end
```

Programming Shortcuts

`Ctrl + R`	`(⌘/ mac)`	comment
`Ctrl + T`	`(⌘T mac)`	uncomment
`Ctrl + {`	`(⌘[ mac)`	promote indent
`Ctrl + }`	`(⌘] mac)`	demote indent
`Ctrl + s`	`(⌘s mac)`	save script
`Ctrl + f`	`(⌘f mac)`	find and replace

2D Plots

Basic Plot Example

Plot sin(x) using 200 points on the domain $0 \leq x \leq 2\pi$. Connects points with a blue line. Label x and y axes and give the plot a title.

```
x = linspace(0,2*pi,200);
y = sin(x);
plot(x,y,'b-');
xlabel('x')
ylabel('y')
title('Graph of sin(x)')
```

Plot Options - Marker and Line Styles

```
plot(x,y,'b-')      % plots a blue line
```

Replace `'-'` with the following to change the plot style:

```
'o'  = circle       '-'  = solid line
'+'  = plus         ':'  = dotted line
's'  = square       '—'  = dashed line
'*'  = star         '-.' = dash-dot line
'x'  = x
'.'  = dot
```

Plot Options - Colors

Replace `'b'` in the above example, with the following to change the color:

```
'r'  = red          'c'  = cyan
'g'  = green        'm'  = magenta
'b'  = blue         'y'  = yellow
'k'  = black        'w'  = white
```

Plot Options - Custom Colors

The color is set by specifying the red R, blue B and green G values, each defined on [0,1]:

```
plot(x,y,'o','Color', [R G B])
```

This example uses dark green circle markers `'o'`:

```
plot(x,y,'o','Color', [0 0.2 0])
```

Plot Options - Line or Marker Thickness

The following code draws a thicker line:

```
plot(x,y,'b-','LineWidth', 2)
```

Thinner line:

```
plot(x,y,'b-','LineWidth', 0.5)
```

Plot Options - Filled Markers

The following draws filled-in circular blue markers:

```
plot(x,y,'bo','MarkerFaceColor', 'b')
```

Shade Under Curve

Fill under curve with red

```
area(x,y,'FaceColor','r')
```

Fill under curve with pink

```
area(x,y,'FaceColor',[1,.8,.8])
```

Axis Limits

Sets the limits along the x and y axes:

```
xlim([xmin xmax])
ylim([ymin ymax])
```

Custom Tick Mark Spacing

Label tick marks from `xmin` to `xmax` in steps `dx` along the x axis. Similar for y axis.

```
xticks(xmin:dx:xmax)
yticks(ymin:dy:ymax)
```

Annotations

Displays text `'hello'` at position (x0,y0) on plot:

```
text(x0, y0, 'hello');
```

Multiple Curves on a Plot

Use `'hold on'` after first plot to prevent subsequent plots from overwriting it:

```
plot(x1,y1)
hold on
plot(x2,y2)
plot(x3,y3)
```

Subplots

The `subplot(ny,nx,n)` command creates a grid of `nx` plot rows and `ny` plot columns and makes the n^{th} plot active. The following creates space for 6 subplots (3 rows, 2 columns) and selects the top right subplot:

```
subplot(3,2,2)
```

Equal Axis Scaling

Set the scaling to be the same for each axis.

```
axis equal
```

Error Bars

To draw error bars given by `yerr` along the y axis, replace the `plot(x,y,'bo')` command with:

```
errorbar(x,y,yerr,'bo')
```

To draw horizontal error bars given by `xerr` use:

```
errorbar(x,y,xerr,'horizontal','bo')
```

Legend

This example plots two curves and displays a legend, labeling the curves “curve 1” and “curve2”.

```
plot(x1,y1)
hold on
plot(x2,y2)
legend('curve 1', 'curve 2')
```

Specialty Plots and Animation

Matlab Plot Gallery

The Matlab website has LOTS of plot examples:
https://www.mathworks.com/products/matlab/plot-gallery.html

Bar Chart

Plots two variables 'val1' and 'val2' on a bar chart:

```
n = 1:5;
val1 = [1 4 2 6 3];
val2 = [2 6 1 4 4];
bar(n,[val1' val2'])
legend('val1', 'val2')
```

Log Plots

These example shows how to logarithmically scale plot axes:

`semilogx(x,y,'b-')`	plots y vs. log(x)
`semilogx(x,y,'b-')`	plots log(y) vs. x
`loglog(x,y,'b-')`	plots log(y) vs. log(x)
`grid on`	turns on grid lines

Polar Plot

This example shows how to plot $r(\theta)$ on a polar plot. The arrays `r` and `theta` store the r and θ data.

```
polar(theta,r,'b-')
```

Parametric Plots

The variables x and y are calculated from a third variable t:

```
t = linspace(0,2*pi,200);
x = sin(2*t);
y = sin(3*t);
plot(x,y)
```

Curve in 3D space

The variables x, y, z are calculated from a third variable t:

```
t = linspace(0,2*pi,200);
x = sin(2*t);
y = sin(3*t);
z = sin(4*t);
plot3(x,y,z)
```

Surface Plot

Plots a function $f(x,y) = 2e^{-((x-.5)^2+y^2} - 2e^{-((x+.5)^2+y^2}$
The `meshgrid()` function creates an x-y grid to evaluate the function. The `surf()` command plots the surface plot.

```
points = linspace(-2, 2, 40);
[X, Y] = meshgrid(points, points);
Z = 2./exp((X-.5).^2+Y.^2)-2./ ...
    exp((X+.5).^2+Y.^2);
surf(X, Y, Z)
```

Create a Video Animation

Creates a video animation of a particle moving according to the parametric equation $(x,y) = (\sin(t), \sin(2t))$. The video file will be 400×400 pixels and will be called "my_video.mp4".

```
hFig = figure('Position',[0 0  400 400]);
set(gca,'nextplot','replacechildren');
v = VideoWriter('my_video','MPEG-4');
open(v);

for t = 0:0.02:2*pi

    x = sin(t);
    y = sin(2*t);

    plot(x,y,'ro', 'MarkerFaceColor','r')
    xlim([-2 2])
    ylim([-2 2])

    frame = getframe(gcf);
    writeVideo(v,frame);

end

close(v);
```

Reading a Data File

If the data file contains only columns of number and each column has the same number of rows, then use the `load()` command to read the contents into a matrix we call `data.` Individual columns may then be extracted from the data matrix:

```
data = load('file.txt');
a    = data(:,1);        % column 1
b    = data(:,2);        % column 2
```

If the data file contains a mixture of numbers and text, then use the `textread()` command. Here's an example with three columns: the first is text, the second two are numbers. In this case, the data are read directly into the individual arrays `txt`, `a` and `b`.

```
[txt,a,b] = textread('file.txt','%s %f %f');
```

This example shows how to skip 3 header lines:

```
[txt,a,b] = textread('file.txt',. . .
   '%s %f %f','headerlines', 3);
```

Writing a Data File

This example shows how to write two arrays to a data file:

```
fileID = fopen('file.txt','w');
for i = 1:length(x)
   fprintf(fileID, '%g %g\n',x(i),y(i));
end
fclose(fileID);
```

`for` Loops

Print numbers 1 through 10

```
for i=1:10
  fprintf('%i\n',i)
end
```

Create a row vector `x` with 10 elements. Each element equals the square of the column number. (Example 1)

```
N = 10;
x = zeros(1,N)
for i=1:N
  x(i) = i^2;
end
```

Add up all elements in vector `x`. (Example 2)

```
s = 0
for i=1:length(x)
  s = s + x(i);
end
fprintf('sum = %g\n', s)
```

Vectorized Loops

Vectorized version of example 1:

```
x = (1:10).^2;
```

Vectorized version of example 2:

```
s = sum(x);
fprintf('sum = %g\n', s)
```

`rand()`	random number drawn from a uniform distribution on [0,1)
`randn()`	random number drawn from a Gaussian distribution ($\bar{x} = 0, \sigma = 1$)
`randi(n)`	random integer between 1 and n

Other Functions

`besselj(x)`	Bessel function of 1st kind
`besseli(x)`	Modified Bessel function
`bessely(x)`	Bessel function of 2nd kind
`besselh(x)`	Bessel function of 3nd kind
`erf(x)`	Error function
`erfinv(x)`	Inverse error function
`airy(x)`	Airy function
`gamma(x)`	gamma function
`factoral(x)`	factorial
`expint(x)`	exponential integral function