Chapter 5. Loops

```
import numpy as np

me = 9.11e-31      # mass of electron
c  = 299792458     # speed of light

u  = 0.1 * c       # particle velocity

gamma = 1 / np.sqrt(1-(u/c)**2)   # gamma factor

KE = (gamma-1) * me * c**2        # relativistic kinetic energy
```

Python for Physicists

for Loops

For loops iterate over a collection of items:

```
for number in [2,3,5]:        # for loop over the collection [2,3,5]
    print("my number = ",number)            # body of the for loop
```

Output:

```
my number = 2
my number = 3
my number = 5
```

Statements in body of loop must be indented by same amount

```
primes = [2, 3, 5, 7, 11]       # define a list of prime numbers
print(" prime  square   cube") # print a header labeling each data column
for p in primes:                # for loop, p will loop over elements of primes
    squared = p**2              # calculate square of p
    cubed   = p**3              # calculate cube of p
    print(f"    {p:2d}     {squared:3d}   {cubed:4d}")
print("done")
```

Output:

```
 prime  square   cube
     2       4      8
     3       9     27
     5      25    125
     7      49    343
    11     121   1331
```

Loop over list of non-numeric items

```
word_list = ['top','quark','gravity','radiation', 'electromagnetic','pion']

for word in word_list:
    print(f"               {word:^15}")
```

Output:

```
      top
     quark
    gravity
   radiation
electromagnetic
     pion
```

Use range(start,stop,step) to iterate over integers

```
for n in range(5):
    print("n =",n)
```

```
n = 0
n = 1
n = 2
n = 3
n = 4
```

```
for n in range(3,8):
    print("n =",n)
```

```
n = 3
n = 4
n = 5
n = 6
n = 7
```

```
for n in range(0,10,2):
    print("n =",n)
```

```
n = 0
n = 2
n = 4
n = 6
n = 8
```

Use enumerate() to return value and index

```
primes = [2, 3, 5, 7, 11]                  # define a list of prime numbers
for i,p in enumerate(primes):              # for loop, i = array imdex
    print(f"index = {i}  prime = {p}")     # print index and value
```

Output:

```
index = 0  prime = 2
index = 1  prime = 3
index = 2  prime = 5
index = 3  prime = 7
index = 4  prime = 11
```

enumerate() example

```
A = [2,5,8,3.14,'a','b']        # define list containing multiple data types

int_type = []                   # int_type will be a list containing indexes of
                                # integers in the array. Initialize to empty list

for i,a in enumerate(A):        # loop over elements in list A
    if type(a) is int:          # check to see if the element is an integer
        int_type.append(i)      # if it is, add the index to our list
print("list of indices containing integer data:  ",int_type)
```

Output: `list of indices containing integer data:  [0, 1, 2]`

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 = []                          # set kinetic energy list to empty list
for m,v in zip(mass, vel):       # loop over both mass and velocity lists
    KE.append(0.5*m*v**2)        # calculate and add KE to KE list
    print(f"m = {m}   v = {v}   KE = {KE[-1]:5.2f}")
```

Output:

```
m = 0.1   v = 2.3   KE =  0.26
m = 0.5   v = 1.2   KE =  0.36
m = 0.9   v = 3.6   KE =  5.83
m = 2.3   v = 5.5   KE = 34.79
```

Accumulator Pattern:

- The Accumulator Pattern consists of a loop and an accumulator variable.
- On each iteration of the loop, the accumulator variable "accumulates" or "gathers" information from the loop

Accumulator Example: Summing integers 1 to 10

- The Accumulator Pattern consists of a loop and an accumulator variable.
- On each iteration of the loop, the accumulator variable "accumulates" or "gathers" information from the loop

```
N = 10                          # N = upper limit of sum

total = 0                       # total = accumulator = sum of integers
for i in range(1,N+1):          # loop over integers 1 to N
    total = total + i           # add i to the running total

print("sum of integers from 1 to",N,"is",total)
```

Output:

```
sum of integers from 1 to 10 is 55
```

Accumulator Example: Extending a list

- Create a list where each item in the list is twice the value of the previous item, starting with 1

```
my_list = [1]                          # initialize the list with 1

for n in range(10):                    # Loop 10 items
    my_list.append(my_list[-1]*2)      # next item = previous item * 2

print(my_list)
```

Output: `[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]`

Nested Loops

```
for row in range(0,100,10):                    # loop over rows
    for col in range(1,11):                    # loop over columns
        product = row + col                    # sum of the row and col
        print(f"{product:3d}  ",end="")        # print with no line feed
    print()                                    # start a new line after
```

Output:

```
  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  30
 31  32  33  34  35  36  37  38  39  40
 41  42  43  44  45  46  47  48  49  50
 51  52  53  54  55  56  57  58  59  60
 61  62  63  64  65  66  67  68  69  70
 71  72  73  74  75  76  77  78  79  80
 81  82  83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98  99 100
```

While Loops

While loops iterate while some condition remains true:

Example: print all powers of 2 less than 100

- We know $2^0 = 1$, so we'll initialize our first power "p" as 1
- Inside our loop, we'll keep multiplying the previous value by 2 to get the next higher power
- Notice we must do the multiplication **after** we print the power. Why is this?

```
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
```

Output:

```
1
4
9
16
25
64
```

List Comprehension

List comprehension is a “fancy” way of combining for loops and lists on a single line of code

Here’s the pattern:

```
[ expression for item in iterable if condition ]
```

Here’s an example of creating a list of squares from 0 to 9 :

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

Output: `[1, 4, 9, 16, 25, 64, 81]`