The flow of control of an algorithm can be completely defined with only four fundamental design structures. These structures are specified using flowcharts and/or pseudo-code notations. The design structures are:

1. **Sequence**, describes a sequence of operations.
2. **Selection**, this part of the algorithm takes a decision and selects one of several alternate paths of flow of actions. This structure is also known as alternation or conditional branch.
3. **Repetition**, this part of the algorithm has sequence of steps that are to be executed zero, one, or more times.
4. **Input-output**, the values of variables are read from an input device (such as the keyboard) or the values of the variables (results) are written to an output device (such as the screen)
Figure: Flow of control.
Flowchart Symbols

Start

Processing

Stop

Condition

True

False

Input/Output
Flowchart with Selection

True

len > 0

False

add 3 to k

decrement k
Flowchart with Input/Output

Input/Output
>>> q = input ("Enter the value of q: ")
Enter the value of q: 45.32
>>> q
45.32
The following example in interactive mode reads two values separated by a space, assigns these to variables \( x \) and \( y \), then converts each to the appropriate type.

```python
>>> x, y = raw_input().split()
12 36.8
>>> x = int(x)
>>> x
12
>>> y = float(y)
>>> y
36.8
```
Example with Selection

```
len > 0

True
add 3 to k

False
decrement k
```
The following example evaluates the condition $len > 0$, to select which operation is to be performed on variable $k$. In Python, this example is written as:

```python
if (len > 0) :
    k= k + 3
else :
    k=k - 1
```
The following example is a selection statement in Python that includes a compound condition.

```python
if a < b or x >= y :
    a = x + 23.45
else :
    a = y
```
A quadratic equation is a simple mathematical model of a second-degree equation and its solution involves complex numbers.

The goal of the solution to the problem is to compute the two roots of the equation.

The mathematical model is:

$$ax^2 + bx + c = 0$$

The given data for this problem are the values of the coefficients of the quadratic equation: $a$, $b$, and $c$.

Because this mathematical model is a second degree equation, the solution consists of the value of two roots: $x_1$ and $x_2$. 
High-level flowchart for solving a quadratic equation.

1. Start
2. Input values of a, b, c
3. Compute discriminant
4. Discriminant < 0
   - Compute real roots
   - Compute complex roots
5. Discriminant ≥ 0
   - Output x1, x2
   - End
Solving A Quadratic Equation with Python

1 # Program : solquad.py
3 # Author : Jose M Garrido, May 18 2016.
4 # Description : Compute the roots of a quadratic equation.
5 # Read the value of the coefficients: a, b, and c
6 # from the input console, display value of roots.
7
8 from math import *
9
10 a = input ("Enter value of coefficient a: ")
11 print "Value of a: ", a
12 b = input ("Enter value of coefficient b: ")
13 print "Value of b: ", b
14 c = input ("Enter value of coefficient c: ")
15 print "Value of c: ", c
16
17 disc = b ** 2 - 4.0 * a * c
18 print "discriminant: ", disc
19 if (disc < 0.0) :
20     # complex roots
21     disc = -disc
22     x1r = -b/(2.0 * a)
23     x1i = sqrt(disc)/(2.0 * a)
24     x2r = x1r
25     x2i = -x1i
26     print "Complex roots "
27     # print "x1r: ", x1r, " x1i: ", x1i
28     x1 = complex(x1r, x1i)
29     #print "x2r: ", x2r, " x2i: ", x2i
30     x2 = complex(x2r, x2i)
31     print "x1: ", x1
32     print "x2: ", x2
33 else :
34    # real roots
35    x1r = (-b + sqrt(disc))/(2.0 * a)
36    x2r = (-b - sqrt(disc))/(2.0 * a)
37    print "Real roots:"
38    print "x1: ", x1r, " x2: ", x2r
Running the Python Program

The following shell commands starts the Python interpreter and it processes the program `solquadra.py`. The program prompts the user for the three values of the coefficients, calculates the roots, then displays the value of the roots. Note that the roots are complex.

```
$ python solquadra.py
Enter value of coefficient a: 1.25
Value of a: 1.25
Enter value of coefficient b: 2.5
Value of b: 2.5
Enter value of coefficient c: 2.85
Value of c: 2.85
discriminant: -8.0
Complex roots
x1: (-1+1.1313708499j)
x2: (-1-1.1313708499j)
```
print "Testing multi-path selection in a Python script"
y = 4.25
x = 2.55
if y > 15.50 :
    x = x + 1
    print "x: ", x
elif y > 4.5 :
    x = x + 7.85
    print "x: ", x
elif y > 3.85 :
    x = y * 3.25
    print "x: ", x
elif y > 2.98 :
    x = y + z*454.7
    print "x: ", x
else :
    x = y
    print "x: ", x
The following lines of code show the Python implementation of selection. The script is stored in file test2.py.

```python
1 # Script for testing while loop
2 x = 12.35
3 MAX_NUM = 15
4 j = 0
5 sum = 0.0
6 while ( j <= MAX_NUM) :
7     sum = sum + 12.5
8     y = x * 2.5
9     j = j + 3
10 print 'Value of sum: ', sum
```
The problem computes the summation of numeric values inputed from the main input device. The program is stored in file `summa.py`.

```python
# Script for summation of input values a while loop
# Script: summa.py

loop_counter = 0
sum = 0.0  # initial value of accumulator variable
innumber = input( "Type number: ")
while innumber > 1.0 :
    sum = sum + innumber
    loop_counter = loop_counter + 1
print "Value of counter: ", loop_counter
innumber = input( "Enter a number: ")
```
The following output listing shows the shell commands that start the Python interpreter with file *summa.py*.

```bash
$ python summa1.py
Type number: 1.5
Value of counter: 1
Type number: 2.55
Value of counter: 2
Type number: 1.055
Value of counter: 3
Type number: 4.12
Value of counter: 4
Type number: 1.25
Value of counter: 5
Type number: 0.0
Value of sum: 10.475
```
Repeat-Until in Python

In Python, the repeat-until loop is not directly supported by a syntactic construct. However, it can be implemented with a while statement.

```python
1 # Script: test5.py
2 # This script tests a loop counter in a repeat-until implemented with a while statement
3
4 Max_Num = 15  # max number of times to execute
5 loop_counter = 1  # initial value of counter
6 loop_cond = False
7 while not loop_cond :
8     print "Value of counter: ", loop_counter
9     loop_counter = loop_counter + 1
10    loop_cond = loop_counter >= Max_Num  # until true
```
1 # Script: summrep.py
2 # This script computes a summation using a
3 # repeat-until loop with a while statement
4
5 sum = 0.0
6 loop_counter = 0
7 innumber = input( "Enter a number: ") # first number
8 lcond = innumber <= 0.0
9 while not lcond:
10   sum = sum + innumber
11   loop_counter = loop_counter + 1
12   print "Value of counter: ", loop_counter
13 innumber = input( "Enter a number: ")
14   lcond = innumber <= 0.0
15
16 print "Value of sum: ", sum
$ python summrep.py
Enter a number: 4.7
Value of counter: 1
Enter a number: 7.88
Value of counter: 2
Enter a number: 0.8
Value of counter: 3
Enter a number: 2.145
Value of counter: 4
Enter a number: 0.0
Value of sum: 15.525
In Python, the simple use of the for statement uses function \texttt{range}. The first, last, and the increment values of the loop counter are specified. The last value specified is not really included as one of the values of the loop counter. The increment is optional; if not included, its value is 1.

```python
# Script: test6.py
# This script tests a for loop
x = 3.45
num = 10
sum = 0.0
for j in range(1, num):
    sum = sum + 12.5
y = x * 2.5
print "Loop counter: ", j
print "Value of sum: ", sum
print "Value of y: ", y
```
$ python test6.py
Loop counter: 1
Loop counter: 2
Loop counter: 3
Loop counter: 4
Loop counter: 5
Loop counter: 6
Loop counter: 7
Loop counter: 8
Loop counter: 9
Value of sum: 112.5
Value of y: 8.625
The \textit{factorial} operation, denoted by the symbol $!$, can be defined in a general and informal manner as follows:

$$y! = y (y - 1) (y - 2) (y - 3) \ldots 1$$

For example, the factorial of 5 is:

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

A mathematical specification of the factorial function is as follows, for $y \geq 0$:

$$y! = \begin{cases} 1 & \text{when } y = 0 \\ y (y - 1)! & \text{when } y > 0 \end{cases}$$
def mfact(num):
    res = 1
    if num > 0:
        for num in range(num, 1, -1):
            res = res * num
        return res
    elif num == 0:
        return 1
    else:
        return -1

y = input("Enter a number to compute factorial: ")
fy = mfact(y)
print "Factorial is: ", fy
Running Factorial Problem in Python

```bash
$ python factp.py
Enter a number to compute factorial: 5
Factorial is: 120

$ python factp.py
Enter a number to compute factorial: 0
Factorial is: 1

$ python factp.py
Enter a number to compute factorial: 1
Factorial is: 1
```
A very simple program consists of data definitions and a sequence of instructions.

The *script mode* is normally used for writing Python programs. Instructions are written into a text file using an appropriate text editor such as *gedit* on Linux and *Notepad++* on Windows.

The text file with the source code is known as a *script* and has a *py* extension.

An instruction performs a specific manipulation or computation on the data, it is written as a language statement in the program.
print \( \langle \text{data\_list} \rangle \)

print y
print "value of x= ", x
\( \text{var_name} = \text{input (\{ string_lit\})} \)

\[
y = \text{input ("Enter value of y: ")}
\]
The name of this Python script is prog02.py.

# This script computes 75% of the value of y
y = 34.5
print "Initial value of y: ", y
y = y * 0.75
print "Final value of y: ", y
This script is started by invoking the python interpreter with the name of the script, `prog02.py`.

```
$ python prog02.py
Initial value of y: 34.5
Final value of y: 25.875
```
This script computes 75% of the value of y
y = input("Enter initial value of y: ")
print "Initial value of y: ", y
y = y * 0.75
print "Final value of y: ", y
The general syntactical form of a function definition is:

```python
def function_name ( [parameters] ) :
    [ local declarations ]
    [ executable language statements ]
```

The relevant internal documentation of the function definition is described in one or more lines of comments, which begins with the characters ("""") and ends with ("""").

```python
def show_message () :
    """
    This function displays a message on the screen.
    """
    print("Computing data")
```
```python
2 # Program : shmesp.py
3 # Author : Jose M Garrido, May 20 2016.
4 # Description : Define and call a simple function.
5
6 def show_message():
7     
8     This function displays a message
9     on the screen
10     
11     print "Computing results ..... "
12
13 y = input("Enter a number: ")
14 sqy = y * y
15 show_message()
16 print "square of the number is: ", sqy
```