BOOLE	30’S
TURING	40’S
CHOMSKY	50’S
BOHM – IACOPINI	60’S

The alphabet of the machine language is binary.

$2024 = 2 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0$

$11111101000 = 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 =$

$= 1024 + 512 + 256 + 128 + 64 + 32 + 8 = 2024$

INTEGERS	int
RATIONALS	float
TEXT (STRING)	str

a 00000

b 00001

c 00010

….

2024	0
1012	0
506	0
253	1
126	0
63	1
31	1
15	1
7	1
3	1
1	1

The 256 characters of the keyboard are encoded by bytes (8 bits).

The Turing machine is a infinite length tape with a head moving left and right to read and write bits plus a memory changing its state (simplest model defining computability).

Chomsky grammar defines high level languages equivalent to the Turing machine.

Bohm and Iacopini proved such equivalence holds if the language comprises only instructions of the following three types: imperative, conditional and repetitive.

---

print “HELLO” OK

---

print HELLO ERROR

HELLO = “HELLO”

print HELLO OK

$2^3 + 2^3 \times 2 = 24$

```
x = 2**3 + 2**3*2
y = (2**3 + 2**3)*2
z = 2**(3 + 2**3)*2
print x
print y
print z
```

```
24
32
4096
```

y = f(x) function with input x and output y

f is not computable if it maps x into y randomly.

The name function is used today to denote subroutines but the main computer program is a function itself.

Subroutine can be default functions, imported functions or function designed by the programmer.

Imported libraries of functions are called modules today (import math or import pickle, for example).

def is the reserved word to design functions.

Operands		and
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

Operands		or
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Operand	not
FALSE	TRUE
TRUE	FALSE

Syntax: x and y x or y not(x)

Operands		and
0	0	0
0	1	0
1	0	0
1	1	1

Operands		or
0	0	0
0	1	1
1	0	1
1	1	1

Parity

```
if  x % 2:
      print x, “is odd”
else:
      print x, “is even”
```

```
if  not(x % 2):
      print x, “is even”
else:
      print x, “is odd”
```

1 + 2 + 3 + …… + 98 + 99 + 100

1 2 3 …… 98 99 100

100 99 98 …… 3 2 1

101 101 101 …… 101 101 101

1 + 2 + 3 + …+ 98 + 99 + 100 = 100(101)/2 = 5050

1 + 2 + 3 + …… + n - 2 + n - 1 + n

1 2 3 …… n - 2 n - 1 n

n n - 1 n - 2 …… 3 2 1

n + 1 n + 1 n + 1 …… n + 1 n + 1 n + 1

1 + 2 + 3 + …… + n - 2 + n - 1 + n = n(n + 1)/2

```
n = raw_input()
n = int(n)
print n*(n + 1)/2
```

```
sum = 0
i = 1
while i <= 100:
    sum = sum + i
    i = i + 1
print sum
```

```
n = raw_input()
n = int(n)
sum = 0
i = 1
while i <= n:
    sum = sum + i
    i = i + 1
print sum
```

1 x 2 x 3 x ……..x n - 2 x n – 1 x n = n! (n factorial)

```
def factorial(n):
    if n < 0:
        return
    if n == 0:
        return    1
    product = 1
    i = 2
    while i <= n:
        product = product* i
        i = i + 1
    return product

n = raw_input()
print factorial(int(n))
```

```
subject = “PHARMACY”
i = 0
while i < len(subject):
    print subject[i]
    i = i + 1
```

```
P
H
A
R
M
A
C
Y
```

```
for character in “PHARMACY”:
    print character
```

Selection Sort

8 5 2 6 9 3 1 4 0 7

0 5 2 6 9 3 1 4 8 7

0 1 2 6 9 3 5 4 8 7

0 1 2 6 9 3 5 4 8 7

0 1 2 3 9 6 5 4 8 7

0 1 2 3 4 6 5 9 8 7

0 1 2 3 4 5 6 9 8 7

0 1 2 3 4 5 6 9 8 7

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

```
i = 0
while i < len(list):
        min = i
        j = i+1
        while j < len(list):
                if list[j] < list[min]:
                        min = j
                j = j + 1
        temp = list[i]
        list[i] = list[min]
        list[min] = temp
        i = i + 1
f = open(“sorted_numbers.pkl”, “w”)
i = 0
while i < len(list):
        pickle.dump(list[i], f)
        i = i + 1
f.close()
```

```
f = open(“sorted_numbers.pkl”, “w”)
i = 0
while i < len(list):
        min = i
        j = i+1
        while j < len(list):
                if list[j] < list[min]:
                        min = j
                j = j + 1
        temp = list[i]
        list[i] = list[min]
        list[min] = temp
        pickle.dump(list[i], f)
        i = i + 1
f.close()
```