# Practical 1

- Name: Chahat Kalsi
- Roll no.: UE193032
- Class: CSE Section 1 (5th Sem)

## Basic Syntax

### Print Statement

In [1]:
```
print("Hello, World!")
```

```
Hello, World!
```

In [4]:
```
print('first python program for DMA')
print('DMA SEMESTER 6')
```

```
first python program for DMA
DMA SEMESTER 6
```

### Lines and Indentation

**Correctly indented conditional statement syntax**

In [5]:
```
if True:
    print("True")
else:
    print("Not True")
```

```
True
```

In [6]:
```
if 5==11:
    print("5 is equal to 11")
else:
    print("5 is not equal to 11")
```

```
5 is not equal to 11
```

### Multi-Line Statements

To span across lines, use \ at the end

In [9]:
```
sum = 1 + \
2 + \
3
print("sum =", sum)
```

```
sum = 6
```

Containers like lists, dicts, sets etc. can span without the line continuation character ()

In [10]:
```
my_list = ['hello', 'hi', 'bye',
           'good morning', 'good night',
           'good afternoon']
```

## Quotations

Strings can use single('), double(") or triple(''' or """) quotes

In [11]:
```
hello = 'hello'
world = "world"
many_lines = """The triple quotes are used
to span the string across multiple lines."""
```

## Comments

In [12]:
```
# This is a comment
hello = "a string called hello" # a comment
```

## User Input

use input to take use input. optional string argument as prompt.

In [13]:
```
name = input("Enter your name: ")
print(f'Hello, {name}!')
```

```
Enter your name: Chahat Kalsi
Hello, Chahat Kalsi!
```

## Multiple statements on one line

Use semi-colon(;) to seperate

In [14]:
```
print('Hello, World!'); x=5; y=5+x; print(f'x={x}, y={y}')
```

```
Hello, World!
x=5, y=10
```

# Variable Types

In [15]:
```
x=100
y="a string"
z = 500.0

print(x, type(x))
print(y, type(y))
print(z, type(z))
```

```
100 <class 'int'>
a string <class 'str'>
500.0 <class 'float'>
```

## Multiple assignment

In [16]:
```
a=b=c=1
print(a,b,c)
```

```
1 1 1
```

In [17]:
```
a, b, c=1, 2.5, "flower"
print(a, b, c)
```

```
1 2.5 flower
```

## Numeric Types

Integer, Float and Complex

### Integer

In [34]:
```
x=1
print(x)


del x # delete the reference to an object using del keyword
try:
    print(x) # this print statement would now throw an error
except NameError:
    print("Error thrown because reference to x was deleted")
```

```
1
Error thrown because reference to x was deleted
```

In [20]:
```
# storing hex numbers: start with 0x
a=0x25f
# storing oct numbers: start with 0o
b=0o76

print(a, b)
```

```
607 62
```

### Float

In [19]:
```
x=1.2
print(x, type(x))
```

```
1.2 <class 'float'>
```

### Complex

represented by: x + yj

In [20]:
```
x=5+9.7j
print(x, type(x))
```

```
(5+9.7j) <class 'complex'>
```

## Strings

In [29]:
```
hello = 'hello'
print(hello)
print(hello[3])
print(hello[0:5:2])
print(hello[0:2])
print(hello[-3:-1])
print(hello*2)
print(hello+", world!")
```

```
hello
l
hlo
he
ll
hellohello
hello, world!
```

## Lists

Mutable

In [30]:
```
lst = [ 'abcd', 5 , 2.23, 'ck', 70.2 ]
List = [123, 'random_name']

print (lst)
print (lst[0])
print (lst[1:3])
print (lst[2:])
print (List * 2)
print (lst + List)
```

```
['abcd', 5, 2.23, 'ck', 70.2]
abcd
[5, 2.23]
[2.23, 'ck', 70.2]
[123, 'random_name', 123, 'random_name']
['abcd', 5, 2.23, 'ck', 70.2, 123, 'random_name']
```

## Tuples

Are like lists, but immutable (read-only lists)

In [1]:
```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print (tuple)           # Prints complete tuple
print (tuple[0])        # Prints first element of the tuple
print (tuple[1:3])      # Prints elements starting from 2nd till 3rd
print (tuple[2:])       # Prints elements starting from 3rd element
print (tinytuple * 2)   # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple
```

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

## Dictionaries

key-value pairs

In [2]:
```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print (dict['one'])       # Prints value for 'one' key
print (dict[2])           # Prints value for 2 key
print (tinydict)          # Prints complete dictionary
print (tinydict.keys())   # Prints all the keys
print (tinydict.values()) # Prints all the values
```

```
This is one
This is two
{'name': 'john', 'code': 6734, 'dept': 'sales'}
dict_keys(['name', 'code', 'dept'])
dict_values(['john', 6734, 'sales'])
```

## Operators

### Logical Operators

In [3]:

```
if True or False:
    print("yes")
```

```
yes
```

In [4]:

```
if True and False:
    print("yes")
```

In [5]:

```
if not False:
    print("yes")
```

```
yes
```

### Arithmetic Operators

In [11]:

```
a=1
b=2
print(1+2)
print(1-2)
print(1*2)
print(1/2)   # division
print(1//2)  # integer division
print(1**2)  # exponentiation
```

```
3
-1
2
0.5
0
1
```

### Bitwise Operators

In [13]:

```
a=60
b=13
print(a&b) # and
print(a|b) # or
print(a^b) # xor
print(~b) # not
print(a<<2) # left shift
print(a>>2) # right shift
```

```
12
61
```

```
49
-14
240
15
```

## Membership operators

In [15]:

```
a = [i for i in range(5)]
print(1 in a)
print(5 in a)
print(0 in a)
print(0 not in a)
```

```
True
False
True
False
```

## Identity Operators

- Compares the memory locations
- basically, checks if id(x)==id(y)

In [16]:

```
x=1
y=2
print(x is y)
print(x is not y)
```

```
False
True
```

In [21]:

```
a=[i for i in range(10)]
b=a
c=a.copy()
print(a is b)
print(a is c)
```

```
True
False
```

In [24]:

```
import math
math.pi
```

Out[24]:

```
3.141592653589793
```