Conversions

Following cells provide overview of different ways to define, transform and obtain **integers, bytes, strings**. At the end of the JN we define conversion to and from **bits, base64**.

Defining values

Literals directly define values `int`, `bytes`, `str`.

In [69]:
```
print("Integers defined with different bases: ", 0b101, 0o101, 101, 0x101, 0X101
print("Bytes: ", b"101", b"\x31\x30\x31")
print("Strings: ", "101", "ABC123", "" )
```

```
Integers defined with different bases: 5 65 101 257 257
Bytes: b'101' b'101'
Strings: 101 ABC123
```

Or you can use array to define bytes.

In [70]:
```
print("Bytes can be also defined by list of ints < 256: ", bytes([49,48,49]))
```

```
Bytes can be also defined by list of ints < 256:  b'101'
```

To int conversions

str to int

In [106…
```
print("int from bits (binary string): ", int('1011', base=2))
print("int from octal string: ", int('1011', base=8))
print("int from decadic string: ", int('1011', base=10))
print("int from hex string: ", int('B', base=16))
```

```
int from bits (binary string):  11
int from octal string:  521
int from decadic string:  1011
int from hex string:  11
```

bytes to int

In [107…
```
print("int from bytes (little endian): ", int.from_bytes([1,2,3,4], 'little'))
print("int from bytes    (big endian): ", int.from_bytes([1,2,3,4], 'big'))
```

```
int from bytes (little endian):  67305985
int from bytes    (big endian):  16909060
```

Little vs Big endian

Difference between little and big endian representation is order of bytes w.r.t. their significance. Little endian index of bytes goes in natural oreder: 0, 1, 2, 3, ... while for big

endian index starts with the last and goes backward: -1, -2, -3, ... 0.

In [108…
```
def my_bytes_to_littlee(byte_array):
    res = 0
    for i in range(len(byte_array)):
        res += byte_array[i]*256**i
    return res

byte_array = [1,2,3,4]

print(my_bytes_to_littlee(byte_array))
print(int.from_bytes(byte_array, 'little'))
```

```
67305985
67305985
```

In [109…
```
def my_bytes_to_int(byte_array):
    res = 0
    for i in range(len(byte_array)):
        res = res*256 + byte_array[i]
    return res

byte_array = [1,2,3,4]

print(my_bytes_to_int(byte_array))
print(int.from_bytes(byte_array, 'big'))
```

```
16909060
16909060
```

To bytes conversions

In [110…
```
print("bytes from hexadecimal string (len must be even): ", bytes.fromhex('ff'))
print("bytes from array(0-255 ints): ", bytes([0, 1, 254, 255]))
print("bytes from int(little): ", int(16909060).to_bytes(length=4, byteorder="li
print("bytes from int(big): ", int(67305985).to_bytes(length=4, byteorder="big")
print("ASCII symbols: ", 'Aa1'.encode('ASCII'))
print("special symbols: ", '₹'.encode('utf-8'))
```

```
bytes from hexadecimal string (len must be even):  b'\xff'
bytes from array(0-255 ints):  b'\x00\x01\xfe\xff'
bytes from int(little):  b'\x04\x03\x02\x01'
bytes from int(big):  b'\x04\x03\x02\x01'
ASCII symbols:  b'Aa1'
special symbols:  b'\xe2\x82\xb9'
```

In [75]:
```
a = 67305985
a.to_bytes(length=4, byteorder="big")
67305985.to_bytes(length=4, byteorder="big") # this doesnt work as 6373.... is n
```

```
  File "/tmp/ipykernel_850/4056206631.py", line 3
    67305985.to_bytes(length=4, byteorder="big") # this doesnt work as 6373....
is not object yet
            ^
SyntaxError: invalid decimal literal
```

To string conversions

In [111…
```
print("hex from integer: ", hex(101))
print("int to string: " , str(101))
print("Bytes with ASCII characters: ", (b'Aa1').decode('ASCII'))
print("Bytes with UTF characters: ", (b'\xe2\x82\xb9').decode('UTF-8'))
```

```
hex from integer:  0x65
int to string:  101
Bytes with ASCII characters:  Aa1
Bytes with UTF characters:  ₹
```

Bits (to and from)

It suffice to convert bits to int and back and then use other conversions.

In [118…
```
print("int from bits (binary string): ", int('1011', base=2))
print("int to 01 string: ", str(bin(11)[2:]))
```

```
int from bits (binary string):  11
int to 01 string:  1011
```

Base64 (to and from)

base64 trabsforms b6-bit blocks to ASCII characters `A-Za-z0-9` with 2 extra `/+`.

In [113…
```
import base64
print("Bytes to base64 characters (bytes): ", base64.b64encode(b'\x01\x02'))
print("base64 string (bytes) to bytes: ", base64.b64decode(b'AQI='))
```

```
Bytes to base64 characters (bytes):  b'AQI='
base64 string (bytes) to bytes:  b'\x01\x02'
```

The padding character `=` represents 2 bits i.e. you can see 0x,1x or 2x padding characters.

In [77]:
```
print(base64.b64encode(b'\x00'))
print(base64.b64encode(b'\x00\x01'))
print(base64.b64encode(b'\x00\x02\x03'))
```

```
b'AA=='
b'AAE='
b'AAID'
```

Task 1: Test correctness of implementation of AES - use following test vectors `pt=00112233445566778899aabbccddeeff`, `key=000102030405060708090a0b0c0d0e0f`, `ct=69c4e0d86a7b0430d8cdb78070b4c55a`. More test vectors can be found here: AES128 test vectors.

In [3]:
```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
```

```
69c4e0d86a7b0430d8cdb78070b4c55a
```

Task 2: Verify that

$$y^2 = x^3 + ax + b \mod p$$

holds for point G = (x, y). First transform strings to byte arrays and then to integers!

EC taken from not official source EC but proper source can be found here SEC1.

In [99]:
```
p = "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF"
a = 'FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFC'
b = '64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1'
G = ('188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012', '07192B95FFC8DA78631011
lhs = 1
rhs = 1
assert lhs == rhs
```