BOOLEAN

In [1]:
```
bool_1=bool(1)
bool_2=bool(0)
bool_3=bool(None)
print(bool_1,bool_2,bool_3)
```

```
True False False
```

CASTING

In [2]:
```
# int()
# str()
# bool()
# float()
# list()
# tuple()
# set()
```

STRING

In [3]:
```
str_="Hello World"
print(type(str))
isinstance(str_,str)
```

```
<class 'type'>
```

Out[3]:
```
True
```

In [4]:
```
str_[0]
```

Out[4]:
```
'H'
```

In [5]:
```
str_[-1]
```

Out[5]:
```
'd'
```

In [6]:
```
# str slicing
print(str_[2:5])  #forward
print(str_[-5:]) #backward
str_[::-1] #reverse
```

```
llo
World
```

Out[6]:
```
'dlroW olleH'
```

CONCATENATION

In [7]:
```
str_1="first"
str_2="second"
print(str_1+" "+str_2)
print(str_*3) #muliple
print(str_1,str_2,sep=":") #sep
print(str_1,str_2,end=" ,") #end
```

```
first second
Hello WorldHello WorldHello World
first:second
first second ,
```

In [8]:
```
# delete string
del str_
print(str_)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 3
      1 # delete string
      2 del str_
----> 3 print(str_)

NameError: name 'str_' is not defined
```

PARTITION

In [9]:
```
str_part="Hello, how are your?, are they?"
print(str_part.partition("are")) #partition the sentence
```

```
('Hello, how ', 'are', ' your?, are they?')
```

In [10]:
```
print(str_part.rpartition("are")) #last
```

```
('Hello, how are your?, ', 'are', ' they?')
```

FUNTIONS

In [11]:
```
str_strip="**********     hi     *******"
print(str_strip.strip("*"),end="")
```

```
     hi     
```

In [12]:
```
print(str_strip.rstrip("*"),end="")
```

```
**********     hi     
```

In [13]:
```
print(str_strip.lstrip("*"),end="")
```
```
     hi    *******
```

In [14]:
```
print(str_part.count("are"))  #count
```
```
2
```

In [15]:
```
str_ex="welcome everyone"
print(str_ex.split())
```
```
['welcome', 'everyone']
```

In [16]:
```
str_ex.find("everyone")
```
Out[16]:
```
8
```

In [17]:
```
str_ex.replace("everyone",",hi")
```
Out[17]:
```
'welcome ,hi'
```

In [18]:
```
str_ex.index("welcome")
```
Out[18]:
```
0
```

In [19]:
```
num_="10"
print(num_.isnumeric(),
num_.isalnum(),
num_.isdecimal(),
num_.isdigit(),
num_.islower(),
num_.isupper(),
num_.isspace(),
num_.isascii())
```
```
True True True True False False False True
```

LIST

In [20]:
```
list_1=[1,2,3,4,5]
list_type=["hi",5.8,7,[5,7,8,9],(7,8,2)]
print(list_type[2])
type(list_type)
print(list_1[:5])
print(list_1[1:4])
print(list_1[-2:])
print(list_1[:-3])
```
```
7
[1, 2, 3, 4, 5]
[2, 3, 4]
[4, 5]
[1, 2]
```

LIST FUNCTION

In [21]:
```
print(list_1.append(10))
print(list_1.insert(0,"hi"))
print(list_1.pop())   #remove last element
print(list_1.pop(5))
del list_1 #.clear()
print(list_1)
```

```
None
None
10
5

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[21], line 6
      4 print(list_1.pop(5))
      5 del list_1 #.clear()
----> 6 print(list_1)

NameError: name 'list_1' is not defined
```

LOOPING & MEMBERSHIP

In [22]:
```
for i in list_type:
    print(i)
```

```
hi
5.8
7
[5, 7, 8, 9]
(7, 8, 2)
```

In [23]:
```
for i in enumerate(list_type):
    print(i)
```

```
(0, 'hi')
(1, 5.8)
(2, 7)
(3, [5, 7, 8, 9])
(4, (7, 8, 2))
```

In [24]:
```
# reverse & sort & sorted
sort_reverse=[3,6,4,57,8,2]
sort_reverse.sort()
print(sort_reverse)
print(sorted(sort_reverse))
sort_reverse.sort(reverse=True)
print(sort_reverse)
```

```
[2, 3, 4, 6, 8, 57]
[2, 3, 4, 6, 8, 57]
[57, 8, 6, 4, 3, 2]
```

In [25]:
```
any(sort_reverse)
```

Out[25]:
```
True
```

In [26]:
```
all(sort_reverse)
```

Out[26]:
```
True
```

TUPLE

In [36]:
```
(1,) #tuple
```

Out[36]:
```
(1,)
```

In [28]:
```
tuple_=(1,2.3,"hi",1)
print(tuple_.index(2.3))
tuple_.count(1)
```

```
1
```

Out[28]:
```
2
```

In [39]:
```
tuple_a=(1,4,6)
tuple_b=(7,4,8)
print(tuple_a+tuple_b)
tuple_x=tuple_a+tuple_b
```

```
(1, 4, 6, 7, 4, 8)
```

In [31]:
```
tuple_a*3
```

Out[31]:
```
(1, 4, 6, 1, 4, 6, 1, 4, 6)
```

In [33]:
```
for i in tuple_a:
    print(i)
```

```
1
4
6
```

```
In [35]: # Asterisk
         ex_tuple=(1,2,3,4,5,4,8,0)
         (x,*y,z)=ex_tuple
         print(x)
         print(y)
         print(z)

         1
         [2, 3, 4, 5, 4, 8]
         0
```

```
In [40]: tuple_x[:]

Out[40]: (1, 4, 6, 7, 4, 8)
```

```
In [43]: print(tuple_x[-5:-2])
         print(tuple_x[1:3])
         tuple_x[::-1]

         (4, 6, 7)
         (4, 6)

Out[43]: (8, 4, 7, 6, 4, 1)
```

```
In [45]: # tuple is immutable (unchangeable)
         # one of the way to update
         tuple_1=(88,)
         tuple_x+=tuple_1
         print(tuple_x)

         (1, 4, 6, 7, 4, 8, 1, 2.3, 'hi', 1, 88)
```

SET

```
In [ ]: # set is non duplicate mutable data type
```

```
In [47]: set_={1,5,8,4,5,4} #duplicate are removed
         set_

Out[47]: {1, 4, 5, 8}
```

```
In [57]: #function
         set_a={1,8,5,7,2,6}
         set_={1,5,8,4,5,4}
```

```
In [58]: print(set_.add(5),
         set_.difference(set_a),
         set_.intersection(set_a),
         set_.union(set_a),
         set_.symmetric_difference(set_a),
         set_.pop(),
         set_.update([5,8,70]),
         set_)
```

```
None {4} {8, 1, 5} {1, 2, 4, 5, 6, 7, 8} {2, 4, 6, 7} 8 None {1, 4, 5, 70,
8}
```

```
In [59]: list(enumerate(set_))
```

```
Out[59]: [(0, 1), (1, 4), (2, 5), (3, 70), (4, 8)]
```

DICT

```
In [60]: dict_={"A":1,"B":2,"C":3}
         dict_
```

```
Out[60]: {'A': 1, 'B': 2, 'C': 3}
```

```
In [61]: dict_.items()
```

```
Out[61]: dict_items([('A', 1), ('B', 2), ('C', 3)])
```

```
In [62]: dict_.values()
```

```
Out[62]: dict_values([1, 2, 3])
```

```
In [64]: dict_.keys()
```

```
Out[64]: dict_keys(['A', 'B', 'C'])
```

```
In [66]: a=[1,4,7,3]
         b={2,5,3,7}
         dict_.fromkeys(a,b)
```

```
Out[66]: {1: {2, 3, 5, 7}, 4: {2, 3, 5, 7}, 7: {2, 3, 5, 7}, 3: {2, 3, 5, 7}}
```

```
In [69]: # dict_.[]
         # dict_.get()            #Access the value
```

```
In [71]: dict_.pop("A")
```

```
Out[71]: 1
```

In [ ]:
```
# (*)args & (**)kwargs
# *args -> variable length Non Keyword Arguments (passed as a tuple)
# **kwargs -> variable length Keyword Arguments (passed as a dictionary)
```