C3 Types and Values

Types

1. type() is built-in function to print the type of a value or variable

```
x = 7
print('x is {}'.format(x))
print(type(x))

#Output: <class 'int'>
```

2. String types
    1. can use 3 quotes (single or double) to put the string in multiple line which including the space between it → no need to separate the print() function in multiple line

```
x = '''

seven

'''
print('x is {}'.format(x))
print(type(x))
```

    2. string is an object because we can format it by calling the method

```
#capitalize is a method
x = 'seven'.capitalize()
print('x is {}'.format(x))
print(type(x))
```

    3. can specify the positional arguments in the format method by using index number

```
x = 'seven {1} {0}'.format(8,9)
print('x is {}'.format(x))
print(type(x))

#Output:
#x is seven 9 8 (index number of 8 is 0 and index number of 9 is 1)
#<class 'str'>
```

    4. can specify the positional arguments in the format method by using index number and operators

```
x = 'seven {1:<9}{0:>9}'.format(8,9)
print('x is {}'.format(x))
print(type(x))

#Output:
#x is seven 9                8 (there are 16 between 8 and 9 as there are 8 spaces from index 1 and also index 0)
#<class 'str'>
```

    5. can put the leading zero, the numbers of zero depends on the space has been defined in {} the zero will be matched between each other

```
x = 'seven "{1:<09}" "{0:>09}"'.format(8,9)
print('x is {}'.format(x))
print(type(x))

#Output:
#x is seven "900000000" "000000008"
#<class 'str'>
```

    6. use f string to replace .format(), which only available after python 3.6

```
a = 8
b = 9
x = f'seven {a} {b}' #need to defined the location of the variable
print('x is {}'.format(x))
print(type(x))
```

3. Numeric types
    1. get the answer in float

```
x = 7 / 3
print('x is {}'.format(x))
print(type(x))
```

    2. get the answer that able to fit between division of 7 and 3

```
x = 7 // 3
print('x is {}'.format(x))
print(type(x))

#Output:
#x is 2
#<class 'int'>
```

    3. get the remainder by using modulo

```
x = 7 % 3
print('x is {}'.format(x))
print(type(x))
```

    4. why the answer is not equal to zero because of the precision of the floating point processor inside the computer but it is not correct

```
from decimal import * # to solve it by import the modules

a = Decimal('.10')
b = Decimal('.30')
x = a + a + a - b
print('x is {}'.format(x))
print(type(x))
```

4. Boolean types (used for logical values and expressions)
    1. return bool results

```
x = 7 > 5
print('x is {}'.format(x))
print(type(x))
```

    2. return none type - used for absence value

```
x = None
print('x is {}'.format(x))
print(type(x))
```

    3. True and False conditions

```
#the results will be False if there is 0 , empty string and None type
x = 0
print('x is {}'.format(x))
print(type(x))

if x:
    print("True")
```

```
else:
    print("False")
```

5. Tuple
    1. able to accept different types of data

```
#there is a list [4, 'four] in the tuple
x = (1, 'two', 3.0 , [4, 'four'], 5)
print('x is {}'.format(x))
print(type(x))
```

    2. id() return a unique identifier for each object

```
x = (1, 'two', 3.0 , [4, 'four'], 5)
y = (1, 'two', 3.0 , [4, 'four'], 5)
print('x is {}'.format(x))
print(type(x))
print(id(x))
print(id(y))
```

    3. return the same id as there are the same object

```
x = (1, 'two', 3.0 , [4, 'four'], 5)
y = (1, 'two', 3.0 , [4, 'four'], 5)
print('x is {}'.format(x))
print(type(x))
print(id(x[0]))
print(id(y[0]))

#Output:
#4384362672
#4383925072
#both are return the same id because there are the same object so that the system will not separate them as different c
```

    4. checking whether x[0] and y[0] is the same object by using "is"

```
if x[0] is y[0]:
    print("Yes")
else:
    print("No")
```

    5. determine the type of the object by using isinstance

```
x = (1, 'two', 3.0 , [4, 'four'], 5)

if isinstance(x, tuple):
    print("Yes, it is tuple")
elif isinstance(x, list):
    print("Yes, it is list")
else:
    print("No")
```

Sequence

1. the list is mutable which means it is changeable

```
#for loop is sequencing through the list
x = [ 1, 2, 3, 4, 5 ]
for i in x:
    print('i is {}'.format(i))
```

2. reassign the one of the element in x list

```
x = [ 1, 2, 3, 4, 5 ]
x[2] = 42
for i in x:
    print('i is {}'.format(i))
```

3. the list with round bracket is known as tuple which is immutable

```
x = ( 1, 2, 3, 4, 5 )
for i in x:
    print('i is {}'.format(i))
```

4. can create the sequence by using range which will end with the last number you specify

```
x = range(5)
for i in x:
    print('i is {}'.format(i))
```

5. can create the starting point and ending point in range

```
x = range(5, 10)
for i in x:
    print('i is {}'.format(i))
```

6. can specify the step by in the third parameter

```
#range(start, end, step)
x = range(5, 50, 5) #will increase 5 for each loop
for i in x:
    print('i is {}'.format(i))
```

7. range is immutable which can be solve by adding list() before the range so that it is mutable

```
x = list(range(5))
x[2] = 42
for i in x:
    print('i is {}'.format(i))
```

8. create a searchable dictionary(mutable)

```
#return a tuple of each items with the key and value
x = {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
#indicate that key three has the value of 42
x['three'] = 42
for k, v in x.items():
    print('k: {}, v: {}'.format(k, v))
```