Statements & Expressions

Camlin | Page
Date / /

Expression - It's combination of anything that returns some value.

e.g.

X = y ⇒ assignment
X * y ⇒ Operation
(x, y) ⇒ aggregate value
X ⇒ Single value
True ⇒ constant value
f( ) ⇒ function.

Note In python Every function return some value even in absence of value.

Statement - It is a line of code
In python their is single statement on each line and no use of semicolon is done. But we can use semicolon for multiple statements per line.

White spaces - white spaces are significant in Python they are not ignored as in other languages. this represents Intended blocks which are used instead of braces in python.

Conditionals

```
x = 42
y = 73
if x > y :  <-- Return bool
    Print ('x>y : x is {} and y is {}'.format(x,y))
else if x == y :
    Print ('equals')
else :
    Print ('x<y : x is {} and y is {}'.format(x,y))
```

output

x<y : x is 42 and y is 72

* No Braces
* No Parethesis

Types and values

1) String Type -

for string type we can use any of (' ', " ", ''' ''', """ """) and results are same.

Strings are objects in python hence we can use methods with them.

x = 'seven'.upper()	SEVEN
x = 'seven'.lower()	seven
x = 'seven'.capitalize()	Seven
x = 'seven'{12}{9}'	Seven 12 9
x = 'seven { }{ }'.format(8,9)	seven 8 9
x = 'seven {0}{1}'.format(8,9)	seven 8 9
x = 'seven {1}{0}'.format(8,9)	seven 9 8
x = f 'seven {a}{b}'. where a=1 b=2	seven 1 2
x = f'seven{a:<9}{b:>9}	Seven 1 2
x = f'seven {a:<09}{b:>09}	Seven 100000000 000000002

2) Numeric Type

1) Int

```
x = 7
```

2) float

```
x = 7.0
```

```
x = .1 + .1 + .1 - .3
```

output - 0---e^{-17} something

~~sacrifices accuracy for precision.~~

∴ we do

```
from decimal import *
a = Decimal ('.10')
b = Decimal ('.30')
x = a+a+a-b
```

output - 0.00

Type <class decimal.Decimal>

* While adding 7+7.5 python implicitely convert int to float

3) Boolean (bool type)

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

Output

```
x is false
<class 'bool'>
```

Note ⇒ By default 0, empty string, None are all False.

Camlin Page
Date / /

None type

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

Output

```
x is None
<class 'Nonetype'>
```

4) Sequence Type

1) Mutable Type

1) List

```
x = [1,2,3,4,5]
x[4] = 42
```

⟵ This is common to all data structure

```
for i in x:
    print ('{}'.format(i))
```

output

1
2
3
4
5

```
x[1:5:2]      ⇒ [2,4]      ⇒ initial, last and steps.
x.index('3')  ⇒ 4          ⇒ Element by index.
x.insert(0,5) ⇒ [5,1,2,3,4,5]  ⇒ insert at index
x.remove(2)   ⇒ [1,3,4,5]  ⇒ Remove item
x.pop('2')    ⇒ [1,2,4,5]  ⇒ Remove index
len(x)        ⇒ 5          ⇒ length of list
x.apped(4)    ⇒ [1:5:2,4]  ⇒ add at last
```

Note ⇒ Indexing done from 0. as in Java.

```
x.[2:] ⇒ [3,4,5]
x.[:2] ⇒ [1'2]
```

} To print certain items by index.

```
x.reverse()   ⇒ To reverse
x.clear()     ⇒ to clear list
x.copy()      ⇒ to copy list
x.extend(other list)  ⇒ to add another list
```

2) Dictionary

```
x = {'one':1', 'two':2', three':3, 'four':4}

x[three] =43
```

keys and value both can be. of any type or data type. eg. int, str, float

1) Loop for one element (without mutation)

```
for i in x:
    Print ('i is {}'. format(i))
```

output
i is one
i is two
i is three
i is four

2) Loop for two elements (with mutation)

```
for k,v. in x.items():
    Print ('k is {}, v is {}'. format(k,v))
```

output
k is one, v is 1
k is two, v is 2
k is three, v is 3
k is four, v is 4

```
for k in x.keys():
for v in x.values():

x['one'] = 5
```

output

one : 5

Pyth

Camlin Page
Date / /

Sets - sets don't allow copy of elements and produce unique element sequence is unorde

```
X = set (1,2,3,4,5,6,7,8)
Y = set (4,5,6,7,8,9,10,11)
```

set (X-y)	in x but not in y	one element only once and sequence is rearranged and made random.
set (X|y)	in x or y	
set (x&y)	in both	
set (X^y)	x or in y but not in both	

```
x = set (we are here)
x ⇒ (weorh)
    (awerh)
    (hrewa)
```
different random sequences produced.

To add or delete elements in dictionary

```
dict1 = {"one": 1, "two": 2}
```

```
dict1["three"] = 3        ⇒ To add /set value.
del dict1["three"]        ⇒ To delete. by key
dict1.key()               ⇒ To get whole keys
dict1.values()            ⇒ To get whole values
for key, value in dict1.items()  ⇒ Iterate over key & val
dict1.clear()
dict1.update(otherdictionary)  ⇒ To update existing items with new.
```

Camlin Page
Date / /

List comprehension - It's a technique in python

① seq = rang (10)
seq2 = [x * 2 for x in seq]
for i in seq2:
print ('{}'.format(i))

Output
0
2
4
6
8
10
=
18

② Similarly x = [(x**x , x**x**x) for x in seq]
Give squares, cubes.

2) z = [x for x in range(0,101) if x%03 ==0]
print (z)

Output 0, 3, 6 ---

2) Immutable

1) Tuple

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

we get error as we cannot make changes to Tuple it is immutable.

3) Mutable & Immutable Range

1)
```
x = rang(10)
for i in x:
    Print ('{}'.format(i))
```

Output
```
0
1
2
3
4
5
...
9
```

2)
```
x = rang (5,10)
for i in ~~rang~~x:
    Print ('{}'.format(i))
```

Output
```
5
6
7
8
9
```

3)
```
x = range (5,50,5)
for i in x:
    print ('{}'.format(i))
```

Output
```
5
10
15
20
25
30
35
40
45
```

4) mutability & immutability

```
x = range [5)
x[2] = 42
```
gives error
Immutable

```
x = list [range (5)]
x[2] = 42
```
Don't give error
Mutable

Note ⇒

Camlin Page
Date / /

Note ⇒ When we assign value to variable we give reference to variable hence we can use any data type to variable in python.

Note ⇒ Dynamic or Duck typing - python determines itself type of data value called duck typing.

Value	type(x)	id
x = 7	int	
x = 7.00	float	
x = Shubham	str	
x = True	bool	
x = None	Nontype	
x = [1,2,3,4,5]	list	
x = (1,2,3,4,5)	tuple	x[2] ⇒ list
x = [1, two, [four, 5]]	list	x[1] ⇒ str
x = (1, 2.00, 5, [four])	tuple	Id's are different for each variable as they aren't same objects but id's of reference are same
y = (1, 2.00, 5, [four])	tuple	

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

```
1) print (type(x))
2) print (type(x[1]))
3) print (id(x))
4) print (id(y))
5) print (id(x[0]))
6) print (id(y[0]))
7) if x is y
       print(y)
   else
       print(N)
```

8) if x[0] is Y[0]:
    print(Y)
else: print(N)

9) if type(x) == tuple:
    print(Y)
else
    print(N)

10) if isInstance(x, tuple):
    print(Y)
else
    print(N)

Outputs

1) <class tuple>
2) <class str>
3) 4329038160
4) 4329276696
5) 4297641088 } same id hence reference have same id. object variable have different id.
6) 4297641088
7) N
8) Y
9) N
10) Y

Operators

Camlin Page
Date / /

1) Comparison operators (return bool)

1) ==
2) !=
3) <
4) >
5) <=
6) >=

2) Logical operators (return bool)

1) and	x and y	if both x & y
2) or	x or y	if any one of x or y
3) not	not x	invert x

3) Identity operator (return bool)

1) x is y	True if they are same objects
2) x is not y	True if not same objects

4) membership operator

x in y	True if x exists inside y collecti
x not in y	True if x isn't inside y collecti

5) Conditional operator (Ternary)

```
hungry = True
x = 'feed' if hungry else 'don't feed'
print(x)
```

output

feed

Camlin | Page
Date / /

6) Arithmetic operators

			X=5, y=3 for x □ y =
1)	+	Addition	8
2)	-	substraction	2
3)	*	multiply	15
4)	/	Division of float etc	1.6666---
5)	//	Division of Integer	1
6)	%	modulo	2
7)	**	Exponent	
8)	-	unary negative	
9)	+	unary positive	

7) Bitwise operators

1)	&	and
2)	|	or
3)	^	xor
4)	<<	Shift left
5)	>>	Shift right

x = 0x0a
y = 0x02
z = x □ y

Print (f'(hex) x is {x:02x}, y is {y:02x}, z is {z:02x}')
Print (f'(bin) x is {x:08b}, y is {y:08b}, z is {z:08b}')

for &

(hex) x is 0a, y is 02, z is 02
(bin) x is 00001010 y is 00000010 z is 00000010

for |

(hex) x is 0a, y is 02, z is 0a
(bin) x is 00001010; y is 00000010, z is 00001111

for ^ x = 0x0a , y = 0x0f , z = x^y

(hex) x is 0a, y is 0f , z is 05
(bin) x is 00001010 , y is 00001111 , z is 00000101

8) Boolean Operator

1)	and	And
2)	or	Or
3)	not	not
4)	in	value in set
5)	not in	value not in set
6)	is	Same object identity
7)	is not	Not Same identity.

a = True
b = False
x = ('bear','bunny','tree')
y = 'bear'

a and b ⇒ false
a or b ⇒ True
not a ⇒ False
y in x ⇒ True
'tree' in x ⇒ True
y is x[0] ⇒ True

Note ⇒ * Id of y and x[0] is same
* Id of two diff variable are different even if they have same values in their list or tuple But their elements have same id.

Variable

1) variable shouldn't have space, use underscore (_)
2) Variable shouldn't have started with numbers
3) Variable should't have special characters in it.

Classes & Objects

Camlin Page
Date / /

Class - How objects defined, definition of objects.
Object - Instance of class.

```
class Duck :                                    ← defining class
    sound = 'Quack'
    movement = 'walk like duck'

    def quack(self):                            ← self not keyword but tradition
        print(self.sound)                         self gives reference of
                                                  object.
    def walk(self):                             ← methods defined inside class
        print(self.walk)

def main():                                     ← main method
    donald = Duck()                             ← creating object

    donald.quack                                ← calling methods from class
    donald.movement

# we can create instance without using main method.
```

Output

```
Quack
walks like duck
```

self should be first parameter for every method.

Camlin | Page
Date / /

Constructor - It is a special type of method in python. defined as

```
def __init__(self,Parameters):
    self._Parameter1 =
    self._Parameter2 =
    self._Parameter3=
        so on.
```

Initialising the variables

Constructors are used to initialize the Parameters.

```
class Animal():

    def __init__(self,type,name,sound):
        self._type = type
        self._name = name
        self._sound = sound

    def type(self):
        return self._type
    def name(self):
        return self._name
    def sound(self)
        return self._sound

def print_animal(o):
    if not isinstance(o,Animal):
        raise TypeError('Print_animal(): require an Animal')
    print('The {} is named "{}" and says "{}".format-
        (o.type(), o.name(), o.sound()))

def main():
    a0 = Animal('kitten','fluffy','purr')
    a1 = Animal('duck','donald','quack')
    print_animal(a0)
    print_animal(a1)
```

Constructor self is first para. always

* getters to access or to return value called as accessors also
* _ is conventional not compulsion.

getters / accessors returns values

method of class

main method

creating objects and initialising with parameters.

calling the method

Camlin | Page
Date / /

print-animal (Animal('Velociraptor', Veronica, 'hello'))
here we get same result without creating object because function parameters works as assignment in python

output

The kitten is named 'fluffy' and says "rwar"
The duck is named 'donald' and says "quack"
The velociraptor is named 'veronica' and says "hello"

```
class person:                                   ← class
    def __init__(self, name):                   ← constructor
        self.name = name
    def talk (self):                            ← method
        print (f"Hi, I am {self.name}")

John = Person ("Johnsmith")                     ] ← object
Bob = person ("Bob Smith")                      ]
~~print (John.name)~~
John.talk                                       ] → calling method.
Bob.talk.
```

Output

Hi I am John Smith
Hi I am Bob Smith

Camlin | Page
Date / /

ENcapsulation

* When variable stored in constructor they are object variable and they are used only if object produced.

* when variable are stored or given outside any method or constructor they become class variable and not object variables.

* When variables are declared as object variable and not as class variable then they are encapsulated

Inheritance ~~is done~~

* Inheritance of class variables and methods done by following syntax.

```
class Animals():
    def __init__(self, **kwargs)
        self._name = name

    def method1(self, a)
        self._sound = sound

    def method2(self, b)
class dog(Animals):
class cat(Animals):
```

} Here we inherite the methods & variable of class Animals.

```
Class Mammal:                    ← superclass
    def walk(self):              ← method in class
        print("walk")

class Dog(Mammel):               ← subclass
    Pass                         ← Pass keyword means nothing just to keep class filled with something we can't let it empty
class cat(Mammel):
    pass

dog1 = Dog()                     ← creating object of Dog class
dog1.walk                        ← Getting method of superclass to subclass object by Inheritance.
```

output

walk

```
class Animal :
    sound = " "
    def __init__(self,name):
        self.name = name

    def speak(self):
        Print ("{sound} I'm {name}! {sound}".format
                ( name =self.name, sound = self.sou
```

```
class piglet (Animal):
    sound = "oink",

hamlet = piglet("Hamlet")
hamlet.speak()
```

⇒ "Oink!" I'm Hamlet! oink!

te? For inheritance we use "is a" rule.

Functions

Note All functions Returns value

1)
```
def isPrime(n):                          # defining function; function name; parameter
    if n<=1:                             # function body
        return False
    for x in rang(2,n):
        if n%x==0:
            return False
    else;                                # ← else for if not for loop.
        return True

n=5
if isPrime(5):                           # calling the function which returns either true or false
    Print(f'{n} is Prime')
else:
    print(f'{n} isn't prime')
```

Output

```
5 is Prime
```

Local & Global variable

Variables defined inside function are local and can be used or modified inside function only.
also any variable modified inside function then modified changes to that variable only accepted inside function.
To make variable global and used outside, inside funct we make it global as e.g. global x inside function
Outside function no need to define it as global

e.g.
```
x = 10
def function():
    global x = 20
```

Camlin Page
Date / /

2)
```
def main():                          ← main called by if statement
    x = kitten()                     ← kitten called by x
    print (x)                        ← x value returned by kitten function

def kitten():                        ← called by main()
    return 'meow'                    ← returns 'meow' string

if __name__ = '__main__': main()     ← if catches main function if if satisfy
```

Output

meow.

Python doesn't discriminate between procedure & method it return value everytime.

Arguments

1)
```
def main():
    kitten(5)

def kitten(a, b=0)
```

Allowed if b is initialise we have to give only one argument

But if not initialise it will throw error.

2)
```
def main():
    x = 5
    kitten(x)
    print(x)
def kitten(a)
    a = 3
    print ('meow')
    print (a)
```

output meow / 3 / 5

3) Python supports variable arguments.

```
def main():
    kitten('a','b','c','d')        ← no. value can be varied

def kitten(*args):                 ← * signify any no. of variable
    if len(args):                      args not keyword but ~~tradition~~
                                   ← ~~If args are greater than zero~~
        for s in args:                 otherwise else will be
            print s                    executed.
    else: print('meow.')

if __name__ == '__main__': main()
```

output

a
b
c
d

4) keyword arguments

```
def main():
    kitten(Buffy='meow', zilla='grr', Angel='rawr')

def kitten(*kwargs):
    if len(kwargs):
        for k in kwargs:
            print('kitten {} says {}'.format(k, kwargs[k]))
    else: print('meow')

if __name__ == '__main__': main()
```

output

kitten	Buffy	says	meow
kitten	zilla	says	grr
kitten	Angel	says	rawr

```
class piglet:
    name = "Piglet"          ~~Compulsory inside method~~
    def speak(self):
        print ("Oink! I'm {}! Oink!".format(self.name))

hamlet = piglet()
hamlet.speak()
```

⇒ Oink! I'm Piglet! Oink!

✡ Using self in arguments / method parameters allows us to use ~~param~~ attributes of class with (.) dot notation. like self.name etc.

Return Value

No distinguished in bern method & procedure and hence returns values every time and 'None' type in case no return statements.

```
def main():
    x = kitten()
    print(type(x))

def kitten():
    print('meow')
    # return 4

if __name__ == '__main__': main()
```

output

```
<class 'NoneType'>
```

for #

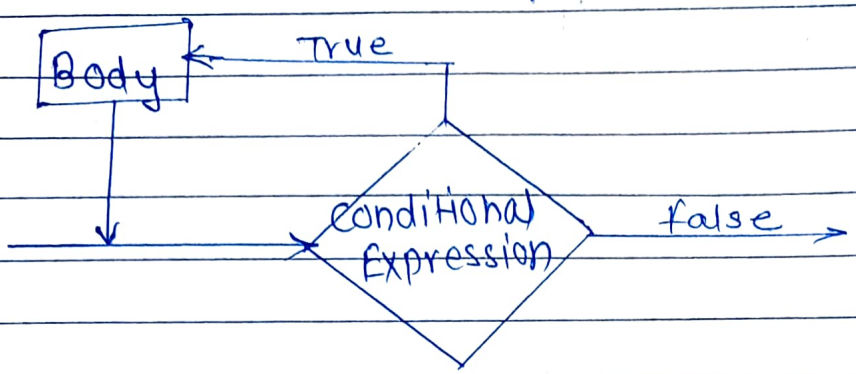
output is

```
<class 'int'>
```

Loops

Camlin | Page
Date / /

1) **While loop** - If conditional Expression true loop continue otherwise breaks.

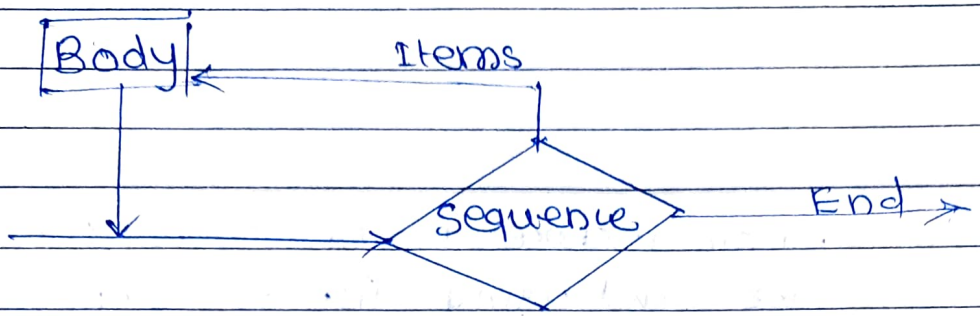



```
Secret = 'Sword fish'
    p = ' '
auth = False
count = 0
max = 5

While   p != Secret :                    ← loop begin, use :
            count += 1                   ← Increment

            if count > max : break       ← to break loop

            if count == 3 : continue     ← to skip element 3
                                           loop will jump to
                                           next

            p = input( f "{count} : password') ← taking input
else:                                      ← else statement
                                             used and executed if
                                             loop breaks due to
                                             condition Expression
                                             unsatisfies.
    auth = True

Print ( " authorised if auth else calling FBI---")
```

Camlin Page
Date / /

Note The common mistake we make is not initializing variables we use or not looking for values or variables we are using properly.

2) For loop - It continues till sequence is executed




```
animals = {'bear', 'bunny', 'dog', 'cat', 'velociraptor'}

for pet in animals:

    if pet == dog : continue        ← It skips dog

    if pet == cat : break           ← It break at cat

    print (pet)

else :                              ← else execute if whole loop is execute on its own.
    print ('that is all animals')
```

I/O

bear
bunny

I/O If we comment the break statement.

bear
bunny
cat
velociraptor
that is all animals.

Camlin | Page
Date / /

Note - For x in range (5):
Print (x)

0
1
2
3
4

Note The common mistake for for loop is not using range for parameter and using list or simple parameters only.

Camlin Page
Date / /

Additional loop controls.

1) if (condition) : break.
if condition satisfy break the loop but breaking of loop causes ignore the else statement (if used) at bottom after loop.

2) if (condition) : continue
if condition satisfy continue causes loop to return at top hence skip that element and jump to next.

3) else :
For while if conditional Expression is not satisfied then executed

Recursion

* function calling itself
e.g. i) people standing in queue and asking person standing in front of them how many peoples are in front of them.
ii) gift of Boxe in Box, Box in Box - - - -

```
def recursive_function(Parameters):
    if base_case_condition(Parameters):
        return base_case_value

    recursive function (modified parameters)
```

Exceptions

Camlin | Page
Date / /

Exception - It's a mechanism to report runtime error

```
def main():                                  ← main function
    try:                                     ← keyword to catch the error at given line
        x = int('foo')        Type of Error
    except ValueError:                       ← catch error
        print('I caught a Error')

If __name__ == '__main__': main()            ← calling main method
```

Catching the error like this gives better handling of Errors we can make changes efficiently.
Instead of getting whole error message we caught the error.

```
def main():
    try:                                     ← To catch the error in function at given line by error message.
        x = 5/0
    except:                                  ← Trying to catch error of unknown type by default.
        print('Unknown error')
    except ValueError:                       ← Trying to catch error of 'ValueError' type only
        print('I caught value error')
    except ZeroDivisionError:                ← Trying catch error of given type i.e. 'Zero Division'
        print('I caught zero division error')
    else:
        print('No error found')              ← If their is no error found this will execute.
        print(x)

if __name__ == '__main__': main()            ← calling main method.
```

output

```
I caught zero division error
```

Camlin Page
Date / /

* In By Default `except:` we can use `print(f'Unknown error:{sys.exc_info()}')` which can give us the type of error. will tell us more about error and specify it.

* Exceptions is the way to report the error from the code which are run time and we can genera the our own type of errors through it.

```
def inclusive_range(*args):          ← Taking unknown no of arguments
    numargs = len(args)              ← assigning no. of argum
    start = 0
    step = 1                         } → initialising.
    if numargs < 1:
        raise TypeError(f'Expected atleast 1 argument got {numargs}')
    elseif numargs == 1:             ← Type error is the object of Exception cla
        stop = args[0]
    elseif numargs == 2:
        (start, stop) = args
    elseif numargs == 3:
        (start, stop, step) = args
    else: raise TypError(f'At most 3 argument, got {num}')
                ↑ keyword to raise error.

    i = start
    while i <= stop:
        yeild i
        i += step
def main():                          ← main method
    for i in inclusive rang(25):     ← for loop 25 is the argument passed
        print(i, end=' ')
    print()
if __name__ == '__main__': main()
```

C

It will give error as arguments are less than expected. hence we can use try to catch the error.

If we change the code as:

```
def main():
    try
        for i in inclusive_range(1,2,3,4):
            Print(i,end=' ')
        Print()
    except TypeError as e:
        Print(f'range error : {e}')

if_name_=='__main__': main()
```

output

rang error : At most 3 argument, got 4.

Hence exceptions are convinient way to report errors from our own class and functions.

String object

Camlin Page
Date / /

Note: `Str` is the string class in python.

As strings are object in python we can use methods on them. like follows.

```
print ('Hello World'.upper())                    HELLO WORLD
      ('Hello World'.lower())                    hello world
      ('Hello World'.swapcase())                 HELLO WORLD
      ('Hello world {}'.format(100*10)           Hello world 1000
      ('Hello
        world'
          {}'.format(10*20)                      Hello
                                                 world
                                                 200
      ('Hello World ß'.casefold())               hello world ss
```

other methods in python

.upper()	→	To upper letters
.lower()	→	To lowecase letters
.capitalize()	→	To capitalize first letter of string
.title()	→	To capitalize first letter of each word
.swapcase()	→	To swap lowes to upper & viceversa
.casefold()	→	Force everything to lower case including special letters. (.lower) won't do this.
.format()	→	to format string with arg. passed in '( ) method we use {} in string as placeholder where variables are printed in string

e.g.
```
x = 21
print ('Hi my age is {}'.format(x))
```

Output

Hi my age is 21

1) we can reverse the position by using placeholder like ---{1}{0}'.format(x,y) here x and y positions interchanged
2) we can also put variable in placeholders as {y}{x} in above case.
3) we can use {x:>5} {x:<5} for space of given no. i.e 5 spaces.
4) we can use {:,} for thousands separation,
5) we use {:3f} for decimal place at given place of digit i.e. 3

Note → strings are immutable and 'id' of every string is different after applying the methods as whole new String is produced.

- .split() - It split string around spaces by default or around character passed in argument. and convert string into 'list' type. output is list.

e.g.

S.split() S=This is my pen ⇒ ['This', 'is', 'my', 'pen']
S.split('i') S=this is my pen ⇒ ['Th', 's', 's my pen']

- .join() - It joins all elements of tuple or list and convert to string separated by space by default or anything passed before join statement and variable of tuple or list is passed as argument in join method.

e.g.

```
l = ['my', 'pen', 'is', 'this']
s = ':'.join(l)    ⇒ 'my:pen:is:this'
s = ' '.join(l)    ⇒ my pen is this
```

Index function

```
pets = "cats & dogs"
print (pet.index(dogs))
⇒ 7
```

Strip function

remove spaces and tabs.

```
" yes ".strip()    ⇒ 'yes'
" yes ".lstrip()   ⇒ 'yes '
" yes ".rstrip()   ⇒ ' yes'
```

characters accessing of strings

```
fruit = Pineapple
print (fruit[0])      ⇒  P
print (fruit[:4])     ⇒  Pine
print (fruit[4:])     ⇒  apple
print (fruit[1,4])    ⇒  ine
```

Creating new string

To replace domain of email with new,

```
def replace_domain(email, old_domain, new_domain):
    if "@" + old_domain in email:
        index=email.index("@"+old_domain)
        new_email = email[:index] + "@"+ new_domain
        return new_email
    return email.
```

Count function

" The number of times e occurs in this string is 4 ".count('e')

⇒ 4

Find with functions

"forest".endswith('rest')

⇒ True

isNumeric function

"12345".isnumeric()

⇒ True.

Camlin Page
Date / /

To find short forms of string.

```
def initials (Phrase):
    words = phrase.split()
    result = " "
    for word in words:
        result = result + word[0]
    return result.upper()

Print ( initials ("Universal serial Bus"))
```

⇒ USB

File I/O

Camlin | Page
Date / /

Opening File

```
def main():
    f = open ('line.txt')
    for line in f:
        Print (line.rStrip())

if __name__ == '__main__': main()
```

keyword to open the file

file name

access each line

rstrip() removes anything after line

open - return file object & read ~~only~~ mode or write mode

rstrip() - strip any wide space or anything after line.

we can use r or w or wr for read, write or readwrite respectively as in arguments.

('filename', 'r') → read mode - by default

('filename', 'w') → write mode

('filename', 'wr') → read writemode.

Camlin | Page
Date / /

Text file

```
def main():
    infile = open('line.txt', 'rt')          ← will open file in read text mode
    outfile = open('line_copy.txt', 'wt')    ← will invoke file for writing txt.
    for line in infile:
        print(line.rstrip(), file = outfile) ←
        Print('.', end = ' ', flush = True)  ← prevent new line after end.
    outfile.close()                          ┐
    infile.close()                           ├→ closing both file
    print('\ndone!')                         ┘

if __name__ == '__main__': main()
```

Output

```
. . . . . .
done
```

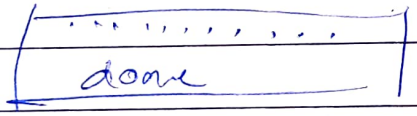
And file is copied into new ~~files~~ files named as in outfile which was not present initially.

Binary file

```
def main():
    infile = open('image1.jpg','rb')
    outfile = open('image1copy.jpg','wb')
    while True:
        buf = infile.read(10240)
        if buf:
            outfile.write(buf)
            print('.',end=' ',flush=True)
        else: break
    outfile.close()
    infile.close()
    print('\n done')

if __name__ == '__main__': main() ← main method.
```

← rb to read buff
← wb to right buff
← Entering loop
← Buffer is of size given in argu
← read Buffer assi value to buf
← If Buffer has value ie. not
← writting Buffer to new file
← Breaking loop wh Buffer empty
← Close files

output



file copied to image1copy.jpg file.

Built in functions

Numeric function

type(x) - to get type of data passed in argument.
int(x) - convert into int type
float(x) - convert into float type
abs(x) - get absolute value i.e. positive.
divmod(x,y) - divide by y and get tuple as ans and Remainder
complex() - To create type data, which gives in terms of j

Camlin Page
Date / /

Container function

X = (1,2,3,4,5) X1 = (6,5,7,8,9)

1) y = len(x) = 5

2) y = max(x) = 5

3) y = min(x) = 1

4) y = list(reversed(x)) = [5,4,3,2,1]

5) y = sum(x) = 15

6) y = sum(x,10) = 15+10 = 25

7) y = any(x) = True ← will return false if every thing is false i.e. 0

8) y = all(x) = True ← will give false if any one is false

9) y = zip(x,x1) ⇒

for a,b in y: print(f'{a}-{b}') ⇒

1-6
2-5
3-7
4-8
5-9

10) x = (cat, dog, rabbit)

for i,v in enumerate(x): print(f'{i}:{v}')

⇒

0 cat
1 dog
2 rabbit

Camlin | Page
Date / /

Objects & class functions

1) isinstance(x, y) ← x is the element/object and y is class
returns bool.

```
X = 42
y = isinstance(x, int)

print(y)
```

output => True

2) type() - gives type
3) id() - gives id.

Camlin Page
Date / /

keywords

False	del	import
True	elif	return
None	else	try
and	except	with
as	finally	yeild
or	for	
assert	while	
Break	from	
class	global	
def	nonlocal	
in	not	
is	pass	
if	raise	
continue	lamba	

Script is small block of codes
.

1) Tkinter
2) Event driven programming.
9. write program to count words in Record.