

Python

Object Oriented Programming

Camlin	Page
Date	/ /

OOP - Powerful paradigm where classes defines concepts and objects are instances of classes.

classes have attributes and methods

Everything we use in python is an object
e.g. string, list, dictionary etc.

* classes - int, str, float etc

+ objects - anything we create instance of class

* methods - functions we use on string after creating instance of string. e.g. upper(), lower() -

Built in functions

dir(" ")

This function used to get information of methods

help(" ") e.g. help(int)

This function used to get information about document

type(" ")

To get the type of string class

Docstring

when we add "''' This is docstring''' " to class, method etc. and use help function as

help(classname) or help(method)

we get DocString which is used to indicating the functioning of that particular class or method.

Just to get information about class & methods

Module - (Python standard library)

We use 'import' keyword to import the python library to code. The abstraction is provided by Python with use of modules.

Python module are separate file that contains classes, functions and other data that allows us to import and make use of these methods and classes in our code.

Essential

Training

Python

Statements & Expressions

Camlin Page

Date / /

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

e.g.

$x=y$ \Rightarrow assignment

$x+y$ \Rightarrow operation

(x,y) \Rightarrow aggregate value

x \Rightarrow single value

True \Rightarrow constant value

f() \Rightarrow function.

Note

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

Statement - It is a line of code

In python there 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 = 72
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 Parenthesis

Types and Values

Camlin	Page
Date	/ /

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{}{}{}'.format(8,9)`

seven 8 9

`x = 'seven{}{}{}'.format(8,9)`

Seven 8 9

`x = f'seven{a}{b}'`

seven 12

where `a=1`
`b=2`

`x = f'seven{a<9}{b>9}'`

Seven 1

`x = f'seven {a:<9}{b:>9}'`

Seven 10000000000000002

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.

\therefore we do

from decimal import *

a = Decimal('1.10')

b = Decimal(' .30')

x = a + a - b

[output = 0.00]

[Type <class 'decimal.Decimal'>]

* While adding 7 + 7.5 python implicitly 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.

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`

`for i in x:`

`print('{}', format(i))`

← This is common to
all data structure

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')` ⇒ [5, 1, 3, 4, 5]

⇒ Remove item

`x.pop('2')` ⇒ [1, 2, 4, 5]

⇒ Remove index

`len(x)` ⇒ 5

⇒ length of list

`x.append(4)` ⇒ [1, 2, 4, 5, 4]

⇒ add at last

Note ⇒ Indexing done from 0. as in Java.

`x[2:]` ⇒ [3, 4, 5]

↳ To print certain items by index.

`x[:2]` ⇒ [1, 2]

↳ To reverse

`x.reverse()`

↳ to clear list

`x.clear()`

↳ to copy list

`x.copy()`

↳ to add another list

`x.extend(otherList)`

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. e.g. int, str, float

1) Loop for one element (without mutation)

`for i in x:`

`print('i is {}'.format(i))`

<u>output</u>
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))`

<u>output</u>
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

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

$X = \text{set}(1, 2, 3, 4, 5, 6, 7, 8)$

$Y = \text{set}(4, 5, 6, 7, 8, 9, 10, 11)$

$\text{set}(X - Y)$ in X but not in Y

$\text{set}(X \cap Y)$ in X or Y

$\text{set}(X \cup Y)$ in both

$\text{set}(X \Delta Y)$ X or in Y but not in both

} one element only once and sequence is rearranged and made random.

$X = \text{set}(\text{we are here})$

$\Rightarrow \{(\text{wearb}), (\text{aworb}), (\text{hrewa})\}$ 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. keys()

⇒ To get whole keys

dict1. values()

⇒ To get whole values

for key, value in dict1. items() ⇒ Iterate over key value

dict1. clear()

dict1. update(other dictionary) ⇒ To update existing items with new.

List comprehension - It's a technique in python

① `seq = range(10)`

`seq2 = [x * 2 for x in seq]`

`for i in seq2:`

`print ('{}' .format(i))`

OUTPUT
0
2
4
6
8
10
12
14
16
18

② Similarly `x = [(x*x, x*x*x) for x in seq]`

give squares, cubes.

③ `z = [x for x in range(0,10) if x%3 == 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) $x = \text{range}(10)$

for i in x:

print ('{}' .format(i))

Output

0

1

2

3

4

5

6

7

8

9

2) $x = \text{range}(5, 10)$

for i in range:

print ('{}' .format(i))

Output

5

6

7

8

9

3) $x = \text{range}(5, 50, 5)$

for i in x:

print ('{}' .format(i))

Output

5

10

15

20

25

30

35

40

45

4) mutability & immutability

$x = \text{range}(5)$

$x[2] = 42$

gives error

immutable

$x = \text{list}(\text{range}(5))$

$x[2] = 42$

Don't give error

mutable

Note =>

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	NoneType	
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
- 6) 4297641088 } object variable have different id.
- 7) No
- 8) T
- 9) N
- 10) Y

Operators

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 anyone 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 collection |
| x not in y | True if x isn't inside y collection |

5) conditional operator (Ternary)

hungry = True

x = 'feed' if hungry else 'don't feed'

print(x)

Output
feed

6) Arithmetic operators

$x=5, y=3$ for $x \square y =$

- 1) + Addition 8
- 2) - Subtraction 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

$x = 0x0a$

$y = 0x02$

$z = x \square y$

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 l

(hex) x is 0a, y is 02, z is 0a

(bin) x is 00001010 ; y is 00000010 . z is 00001111

for A

$$x = 0x0a, y = 0x0f, z = 2xy$$

(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 \Rightarrow false

a or b \Rightarrow True

not a \Rightarrow False

y in x \Rightarrow True

'tree' in x \Rightarrow True

y is x[0] \Rightarrow True

Notes * 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 shouldn'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):

 print(self.sound)

] ← self not keyword but tradition
 self gives reference of
 object.

def walk(self):

 print(self.walk)

] ← methods defined inside class

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 = ] Initialising  
    self._parameter2 = the Variables  
    self._parameter3 = ]  
    so on.
```

constructors are used to initialize the ~~statement~~ Parameters.

class Animal():

```
def __init__(self, type, name, sound): ] constructor  
    self._type = type self is first  
    self._name = name para. always  
    self._sound = sound ]  
    * getters to access  
    or to return value  
    called as accessors also  
    * is conventional  
    not compilation.
```

```
def type(self): ] getters / accessors,  
    return self._type returns values
```

```
def name(self): ] returns values  
    return self._name
```

```
def sound(self): ] returns values  
    return self._sound
```

```
def print_animal(o): ] ← method of class
```

```
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(): ] ← main method
```

```
a0 = Animal('kitten', 'fluffy', 'mew') ] creating objects  
a1 = Animal('duck', 'donald', 'quack') and initialising with  
parameters.
```

```
print_animal(a0) ] ← calling the  
print_animal(a1) method
```

`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 "mew"
 the duck is named 'donald' and says "quack"
 the velociraptor is named 'veronica' and says "hello"

Class person:

`def __init__(self, name):` ← constructor

`self.name = name`
`def talk(self):` ← method

`print(f"Hi, I am {self.name}")`
`John = Person("Johnsmith")`
`Bob = person("Bob Smith")`
`print(John.name)`
`John.talk`
`Bob.talk`

] ← object

] → calling method.

Output

Hi I am John Smith
Hi I am Bob Smith

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

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

```
class Animals():
    pass
```

```
def __init__(self, *args, **kwargs):
    self.name = name
```

```
def method1(self, a):
    self.sound = sound
```

```
def method2(self, b)
```

```
class dog(Animals):
    pass
```

```
class cat(Animals):
    pass
```

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

```
class Mammal:
    pass
```

← Superclass

```
def walk(self):
    print("Walk")
```

```
class Dog(Mammal):
    pass
```

← Subclass

← Pass keyword means nothing just to keep class filled with something we can let it empty

```
class Cat(Mammal):
    pass
```

```
dog1 = Dog()
```

← creating object of Dog class

```
dog1.walk
```

← Calling method of superclass to subclass object by inheritance.

Output

```
walk
```

class Animal :

sound = ""

def __init__(self, name):

self.name = name

def speak(self):

print ("{} I'm {}! {}".format

(name = self.name, sound = self.sound))

class piglet(Animal):

sound = "oink!"

hamlet = piglet("Hamlet")

hamlet.speak()

⇒ "Oink! I'm Hamlet! Oink!"

For inheritance we use "is a" rule.

Functions

Note

All functions Returns value

1) `def isPrime(n)`

↓ defining function ↓ function name ↓ parameter

if $n \leq 1$:
return False

for x in range(2, n):
if $n \% x == 0$:
 return False

else:
 return True

← else for ~~loop~~
not for loop.

$n = 5$ ↓ calling the function which returns
 either true or false

if isPrime(5):
 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 function 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`

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

→ main called by if stat.
 → kitten called by x
 → x value returned by kitten function

6) `def kitten():
 return 'meow'`

→ called by main()
 ← returns 'meow' string

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

← if calls main function if if satisfy

Output

meow.

Python doesn't discriminate between procedure & method it return Value everytime we call it.

Arguments

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

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

`def kitten(a, b=0)` ← But if not initialise it will throw error.

3) `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 traditionally
        for s in args: ← If args are greater than zero
            print(s) ← otherwise else will be
    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 allow us to use param attributes or class with(.) dot notation like self.name etc.

Return Value

No distinguished between method & procedure and hence returns values everytime 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'>
```

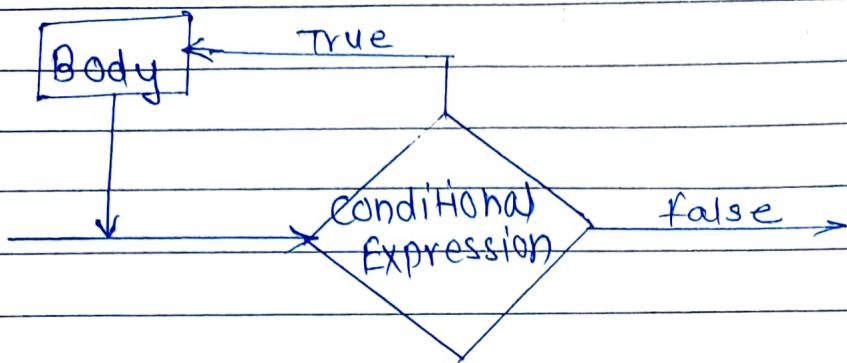
For #

Output is

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

Loops

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



Secret = 'sword fish'

p = ''

auth = False

count = 0

max = 5

while p != Secret :
 count += 1

← loop begin, use !
← 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

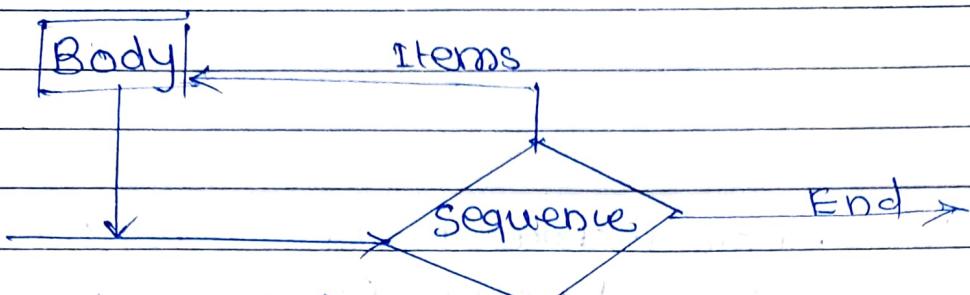
else:
 auth = True
← else statement used and executed if loop breaks due to condition Expression unsatisfies.

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

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 :`

`print ('that is all animals')`

← else execute if whole loop is execute on its own.

I/O

bear
bunny

I/O

If we comment the break statement .

bear
bunny
cat

Velociraptor
that is all animals .

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.

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 Box in Box, Box in Box - - -

def recursive_function(Parameters):

 if base_case_condition(Parameters):
 return base_case_value

recursive function (modified Parameters)

Exceptions

Exception - It's a mechanism to report runtime error

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

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

Output

I caught zero division error

* 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 generate our own type of errors through it.

```
def inclusive_range(*args):
    num_args = len(args) ← Taking unknown no. of arguments
    start = 0 ← assigning no. of arguments
    step = 1 ]→ initialising.

    if num_args < 1:
        raise TypeError(f'Expected at least 1 argument, got {num_args}') ← Type error is the object of Exception class
    elif num_args == 1:
        Stop = args[0]
    elif num_args == 2:
        (start, stop) = args
    elif num_args == 3:
        (start, stop, step) = args
    else:
        raise TypeError(f'At most 3 arguments, got {num_args}') ← keyword to raise error.
```

i = start

while i <= stop:

yield i

i += step

def main():

for i in inclusive_range(25): ← for loop 25 is the argument passed
print(i, end=' ')

print()

If __name__ == '__main__': main()

← main method

← for loop 25 is the argument passed

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=' ')
    except TypeError as e:
        print(f'range error : {e}')
if __name__ == '__main__':
    main()
```

Output

range error : At most 3 argument, got 4.

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

String object

Camlin	Page
Date	/ /

Notes

`str` is the string class in python.

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

<code>print ('Hello World'.upper())</code>	HELLO WORLD
<code>('Hello World'.lower())</code>	hello world
<code>('Hello World'.swapcase())</code>	HEllo wORLD
<code>('Hello world {}'.format(100*10))</code>	Hello world 1000
<code>('Hello\nworld')</code>	Hello world
<code>{} '.format(10*20)</code>	200
<code>('HelloWorld β'.casefold())</code>	hellow world ss

other methods in python

- `.upper()` → To UPPER letters
- `.lower()` → To lowercase letters
- `.capitalize()` → To Capitalize first letter of string
- `.title()` → To Capitalize first letter of each word
- `.swapcase()` → To swap lower 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 instead

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 `{}{}:format(x,y)` here x and y positions interchanged
- 2) we can also put variable in place holders 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.

- 5. `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()` $s = \text{This is my pen} \Rightarrow ['\text{This}', '\text{is}', '\text{my}', '\text{pen}']$

`s.split('i')` $s = \text{this is my pen} \Rightarrow ['\text{Th}', '\text{s}', '\text{s my pen}']$

- `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 or tuple or list is passed as argument in join method.

e.g.

$l = ['\text{my}', '\text{Pen}', '\text{is}', '\text{this}']$

$s = '::'.join(l) \Rightarrow '\text{my: pen: is: this}'$

$s = ' '.join(l) \Rightarrow \text{my pen is this}$

Index function

`pets = "cats & dogs"`

`print(pet.index("dogs"))`

$\Rightarrow 7$

Strip function

remove spaces and tabs.

`" yes ".strip() \Rightarrow 'yes'`

`" yes ".lstrip() \Rightarrow 'yes '`

`" yes ".rstrip() \Rightarrow ' yes'`

characters accessing of strings

fruit = Pineapple

print(fruit[0]) \Rightarrow P

print(fruit[:4]) \Rightarrow Pine

print(fruit[4:]) \Rightarrow apple

print(fruit[1:4]) \Rightarrow 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)

\Rightarrow 4

Endwith functions

"forest".endswith('rest')

\Rightarrow True

isnumeric function

"12345".isnumeric()

\Rightarrow True,

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()
```

Annotations:

- def main(): keyword to open the file
- f = open('line.txt') file name
- for line in f: access each line
- print(line.rstrip()) rstrip() removes anything after line
- if __name__ == '__main__': main()

Open - return file object & read mode or write mode

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

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

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

('filename', 'w') → write mode

('filename', 'wr') → read write mode.

Text file

```
def main():
    infile = open('line.txt', 'rt') ← will open file in
    outfile = open('linecopy.txt', 'wt') ← read text mode
    for line in infile:
        print(line.rstrip(), file=outfile) ← will invoke file
        print(',', end=' ', flush=True) ← prevent newl
    outfile.close() ← closing both file
    infile.close()
    print('done!')
```

```
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') ← rb to read buff.
    outfile = open('image1copy.jpg', 'wb') ← wb to write buff.
    while True:
        buf = infile.read(1024) ← Buffer is of size given in argument.
        if buf: ← read Buffer & assign value to buf.
            outfile.write(buf) ← If Buffer has value ie. not empty.
            print(':', end=' ', flush=True)
        else: break ← writing Buffer to new file.
    outfile.close() ← Breaking loop when Buffer empty.
    infile.close()
    print('\n done')

```

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

Output

.....
done

File copied to image1copy jpg file.

Built in functions

Camlin	Page
Date	1 / 1

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 i

Container function

$x = (1, 2, 3, 4, 5)$ $x_1 = (6, 5, 7, 8, 9)$

1) $y = \text{len}(x)$ = 5

2) $y = \text{max}(x)$ = 5

3) $y = \text{min}(x)$ = 1

4) $y = \text{list}(\text{reversed}(x))$ = $[5, 4, 3, 2, 1]$

5) $y = \text{sum}(x)$ = 15

6) $y = \text{sum}(x, 10)$ = $15 + 10 = 25$

7) $y = \text{any}(x)$ = True ← will return False if everything is false i.e. 0

8) $y = \text{all}(x)$ = True ← will give False if any one is False

9) $y = \text{zip}(x, x_1)$ \Rightarrow

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

* for a, b in y : print(f'{a}-{b}')

10) $x = (\text{cat}, \text{dog}, \text{rabbit})$

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

\Rightarrow

0 cat
1 dog
2 rabbit

Objects & class functions

1) `isinstance(x, y)`

$x = 42$

$y = \text{isinstance}(x, \text{int})$

\leftarrow x is the element/object and y is class
returns bool.

5) `print(y)`

output \Rightarrow True

2) `type()` - gives type

3) `id()` - gives id.

Keywords

false	del	import
true	elif	return
None	else	try
and	except	with
as	finally	yield
or	for	
assert	while	
Break	from	
class	global	
def	nonlocal	
in	not	
is	pass	
if	raise	
continue	lambda	

Script is small block of codes

1) Tkinter

2) Event driven programming,

g. write program to count words in
Record.