Programming with Python

Pythonic Ways

SETS

no-duplicate

```
array = [1, 2, 2, 3, 3, 3]
unique = []
for i in array:
    if i not in unique:
        unique.append(i)
print(unique)
```

```
[1, 2, 3]
```

```
s = {1, 2, 2, 3, 3, 3}
print(s)
print(type(s))
```

```
{1, 2, 3}
<class 'set'>
```

syntax

```
s = { None, 1, 1.68, 'string' }
print(s)
```

```
{None, 1, 'string', 1.68}
```

```
print(set('hello world'))
```

```
{'l', 'e', ' ', 'd', 'h', 'w', 'o', 'r'}
```

methods

```
shapes = set()

shapes.add('hexagon')
print(shapes)

shapes.update({'circle', 'square'})
print(shapes)

shapes.remove('circle')
print(shapes)

try:
    shapes.remove('circle')
except KeyError:
    shapes.discard('circle')  # ignore if not found
```

```
{'hexagon'}
{'square', 'circle', 'hexagon'}
{'square', 'hexagon'}
```

operation

```python
A = {1, 2, 3}
B = {3, 4, 5}

print(A.union(B), A | B)
print(A.intersection(B), A & B)
print(A.difference(B), A - B)

print(A.symmetric_difference(B), A ^ B)  # remove common
```

```
{1, 2, 3, 4, 5} {1, 2, 3, 4, 5}
{3} {3}
{1, 2} {1, 2}
{1, 2, 4, 5} {1, 2, 4, 5}
```

summary

1. sets is unordered list with unique items
2. set theory operations are possible

FILES

I/O (Input and Output)

syntax

file I/O is done through file objects

```
with open('/path/to/file', [mode], encoding='utf8') as fp:
    pass # do some operation here
```

mode	description
'r'	read [Default]
'w'	write
'a'	append
'+'	read + write

writing and reading

for windows `C:\\path\\to\\file`

```
with open('myfile', 'w') as fp:
    fp.write('hello\n')
    fp.write('this file is written from python')

with open('myfile') as fp:
    data = fp.read()
    print(data)
```

```
hello
this file is written from python
```

1. add line number

```
with open('file-read.py') as fp:
    for line in fp:
        print(end=line)
```


```
with open('file-read.py') as fp:
    for line in fp:
        print(end=line)
```


```
1  with open('file-read.py') as fp:
2      for line in fp:
3          print(end=line)
```

2. count

```
s = 'happy\nprogramming\n  \nin\tpython'
print(s.split())
print(s.split('\n'))
print(s.splitlines())
```

```
Frankenstein.txt has 7833 lines
Frankenstein.txt has 78098 words

open(FILE, encoding='utf8') for windows
```

3. search

```
s = 'running'
print('ing' in s)
print(s[:3],  s[:3]  == 'run', s.startswith('run'))
print(s[-3:], s[-3:] == 'ing', s.endswith('ing'))
```

```
24 words has "program"
14 words starting with "program"

# In linux install package "wamerican" or "wordlist" or "words"
# ftp://ftp.gnu.org/gnu/aspell/dict/en
```

4. count 5 letter words

```
with open('words') as fp:
    for line in fp:
        w = line.strip()
        print(w, len(w))
        break
```

```
A 1
```

```
7654
```

5. vowels

count words contain all vowels

931

file pointer

```
with open('myfile') as fp:
    print(fp)
    print(fp.read())
    print(fp.read())
    print('position:', fp.seek(0))
    print('fp.read(bytes=6):', fp.read(6)) # seek
    print(fp.tell())
    print(next(fp))
```

```
<_io.TextIOWrapper name='myfile' mode='r' encoding='UTF-8'>
hello
this file is written from python

position: 0
fp.read(bytes=6): hello

6
this file is written from python
```

summary

1. file are stored as string
2. file is opened as stream

DICTIONARY

syntax

```
print(type({}))  # its not set
print(type(dict()))
```

```
<class 'dict'>
<class 'dict'>
```

```
# can be be of different type except list
print({
    100    : 'integer',
    1.0    : 'float',
    1.5    : 'float',
    1j     : 'complex',
    (1, 2) : 'pair tuple',
    True   : 'bool=true',
    False  : 'bool=false',
    bool   : 'ok this is going insane',
    int    : 'or its just a trailer',
})
```

operation

```
d = {
     'key'   : 'val',
     'first' : 65,
}
print(d)
print(d['key'])

d['new'] = 'adding new things'
d.pop('key')  # remove
print(d)

d.update({1 : 'one', 2 : 'two'})
print(d)
```

```
{'key': 'val', 'first': 65}
val
{'first': 65, 'new': 'adding new things'}
{'first': 65, 'new': 'adding new things', 1: 'one', 2: 'two'}
```

union

```
info = {
    'name'       : 'Dr. Victor Frankenstein',
    'spouse'     : 'Elizabeth Lavenza',
    'occupation' : 'Doctor',
    'nickname'   : 'The Modern Prometheus'
}

print(info | {
    'nickname' : 'Mad Scientist',
})
```

```
{
    "name": "Dr. Victor Frankenstein",
    "spouse": "Elizabeth Lavenza",
    "occupation": "Doctor",
    "nickname": "Mad Scientist"
}
```

1. count letter frequency

```
print(d := dict.fromkeys(['one', 'two'], 1))
d['two'] += 1
print(f"{d['two']=}")
```

```
{'one': 1, 'two': 1}
d['two']=2
```

```
phrase: the quick brown fox jumps over the lazy dog
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 1 1 1 3 1 1 2 1 1 1 1 1 1 4 1 1 2 1 2 2 1 1 1 1 1
```

2. count char frequency

```
print(d := { 1 : 'one' })
print('2 in d:', 2 in d)
d[2] = 'two'
print(f'{d[2]=}')
```

```
2 in d: False
d[2]='two'
```

```
phrase='list of punctuations are .,!;'
l i s t   o f p u n c a r e . , ! ;
1 2 2 3 4 2 1 1 2 2 1 2 1 1 1 1 1 1
```

3. word length distribution

```
d = {}
print(d.get('undefined key'))
print(d.get('undefined key', 'default value'))
```

```
None
default value
```

```
[(1, 52), (2, 491), (3, 1471), (4, 3949), (5, 7654), (6, 12866), (7, 17313
```

4. most frequent word

c style 😒

```
d = { 1 : 'one', 2 : 'two' }
for k in d:
    print(k, d[k])
```

😎 py style

```
d = { 1 : 'one', 2 : 'two' }
for k, v in d.items():
    print(k, v)
```

```
"the" occured 4056 times in "Frankenstein.txt"
```

5. 6 letter word 6 anagrams

```
['capers', 'crapes', 'pacers', 'parsec', 'recaps', 'scrape', 'spacer']
['carets', 'caster', 'caters', 'crates', 'reacts', 'recast', 'traces']
['deltas', 'desalt', 'lasted', 'salted', 'slated', 'staled']
['drapes', 'padres', 'parsed', 'rasped', 'spared', 'spread']
['palest', 'pastel', 'petals', 'plates', 'pleats', 'septal', 'staple']
```

summary

1. dict is pair of key value
2. dict can be look up table

PROJECT 02

convert number to words

```
input the number: 123
words: one hundred and twenty three
```

map

```
0; zero
1; one
..
1000000000000000000; quintillion
1000000000000000000000; sextillion
1000000000000000000000000; septillion
1000000000000000000000000000; octillion
1000000000000000000000000000000; nonillion
```

SEQUENCE

concatenation

```
1 print([1, 2, 3] + [4, 5, 6])
2 print((1, 2, 3) + (4, 5, 6))
3 print('abc' + 'efg')
```

```
[1, 2, 3, 4, 5, 6]
(1, 2, 3, 4, 5, 6)
abcefg
```

repeat

```
1 print('a' * 5)
2 print([0] * 5)
3 print((0,) * 5)
```

```
[0, 0, 0, 0, 0]
(0, 0, 0, 0, 0)
aaaaa
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
```

repeat

```
1 print('a' * 5)
2 print([0] * 5)
3 print((0,) * 5)
```

```
[0, 0, 0, 0, 0]
(0, 0, 0, 0, 0)
aaaaa
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
```

```
1 # also supports groups
2 print([0, 1] * 5)
3 print((1, 2) * 5)
4 print('abc' * 5)
```

```
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
(1, 2, 1, 2, 1, 2, 1, 2, 1, 2)
abcabcabcabcabc
```

search

```
1 print(1 in [1, 2, 3])
2 print(1 in (2, 3, 4))
3 print('a' in 'abc')
4 print('ab' in 'abc')
```

```
True
False
True
```

```
1 a = [10, 20, 30]
2 print(i:=a.index(20), a[i])
3
4 t = (40, 50, 60)
5 print(i:=t.index(60), t[i])
6
7 s = 'hello world'
8 print(i:=s.index('world'), s[i:])
```

```
1 20
2 60
6 world
```

copy

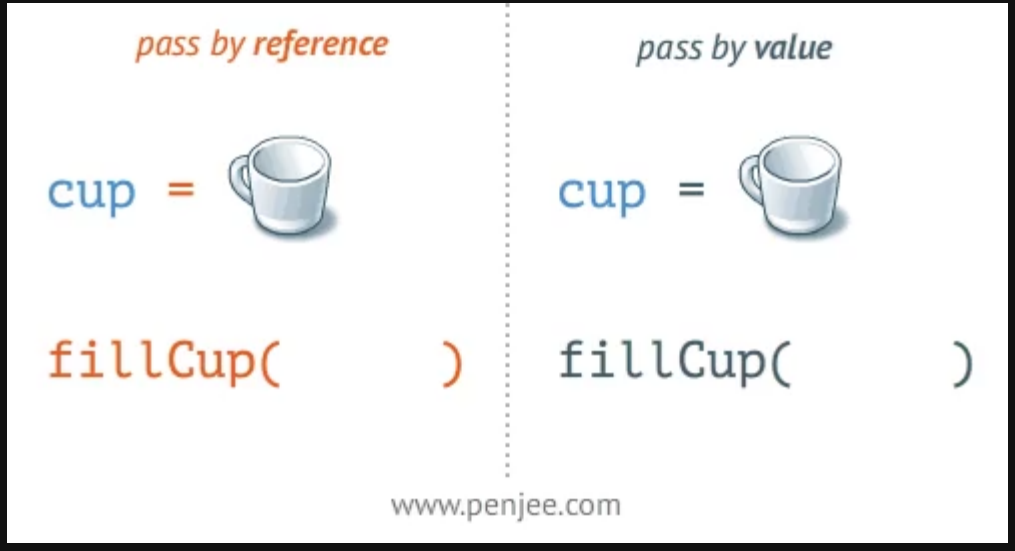
```
1 a = list(range(10))
2 b = a
3 b[0] = 100
4 print(a)
5 print(b)
6 print(id(a), id(b))
```

```
[100, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[100, 1, 2, 3, 4, 5, 6, 7, 8, 9]
126927924845824 126927924845824
```

```
1 a = list(range(10))
2 b = a[:]  # copy
3 c = a.copy()
4 print(id(a), id(b), id(c))
```

```
134919367835200 134919369705920 134919369707712
```

passby


pass by reference
cup =
fillCup( )
pass by value
cup =
fillCup( )
www.penjee.com

loops

```
for i in 1, 2, 3, 4:
    print(i)
```

```
for c in 'hello':
    print(i)
```

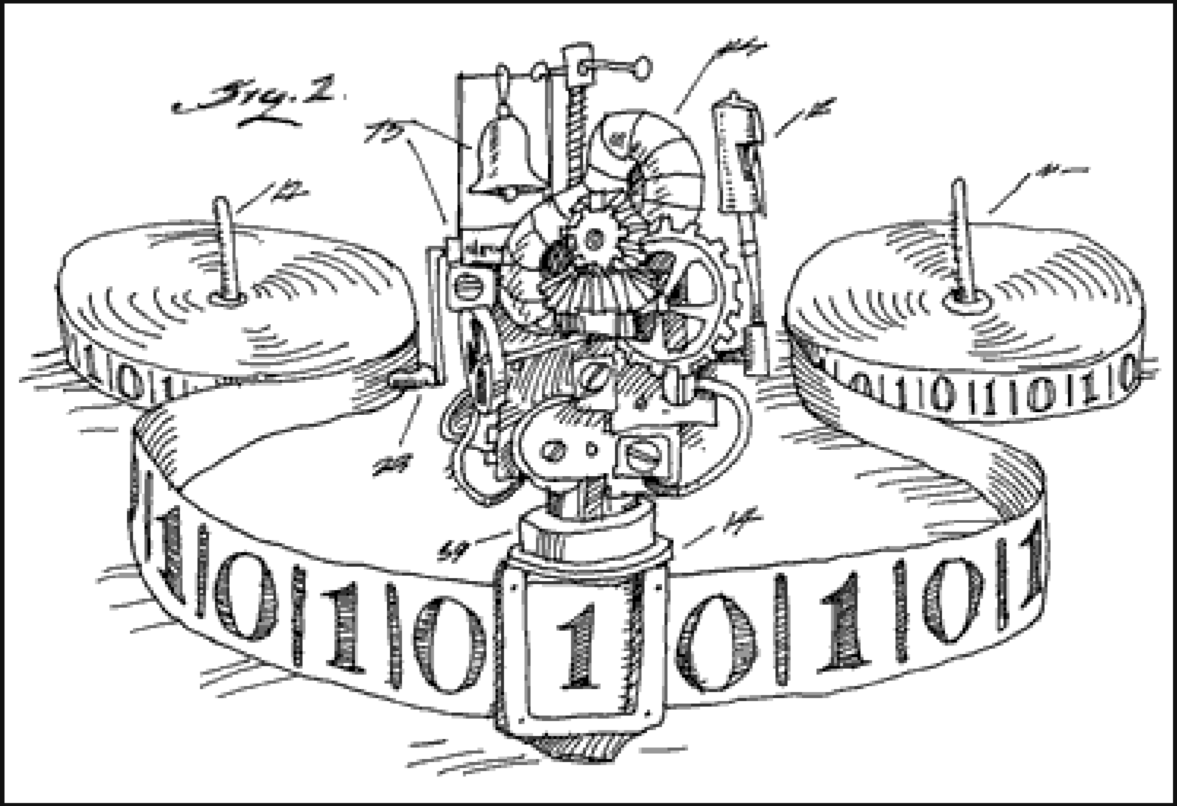
```
d = { 1:10, 2:20, 3:30, 4:40 }
for i in d:  # d.keys()
    print(i)

for k, v in d.items():
    print(k, v)
```

```
1
2
3
4
```

summary

	name	action
+	plus	concatenation
*	asterisk	repeat
[]	get item	get nth item
[:]	slice	get segment, copy
in	in	check item existence
index()	index	return first index of value

ITERATORS



syntax

```
 1 it = iter('py')
 2
 3 print(it)
 4 print(type(it))
 5
 6 print(next(it))
 7 print(next(it))
 8
 9 try:
10     next(it)
11 except StopIteration as e:
12     print('oops!')
```

enumerate

```
for i, c in enumerate('hi'):
    print(i, c)
```

```
0 h
1 i
```

```
print(it := iter('hi'))
print(enum := enumerate(it, 0))

while True:
    try:
        i, c = next(enum)
        print(i, c)
    except StopIteration as e:
        break
```

```
<str_ascii_iterator object at 0x71f8680c3c40>
<enumerate object at 0x71f868608fe0>
0 h
1 i
```

1. undumpify

```
# 8-bit hexdump
print(ord('a'))
print(hex(ord('a')))
```

```
97
0x61

67726561742c20796f752063616e207265616421
```

zip 🤐

```
print(*enumerate('hello'))
print(*zip(range(5), 'hello'))
print(*zip(range(3), 'hello'))
```

```
(0, 'h') (1, 'e') (2, 'l') (3, 'l') (4, 'o')
(0, 'h') (1, 'e') (2, 'l') (3, 'l') (4, 'o')
(0, 'h') (1, 'e') (2, 'l')
```

```
print(dict(zip([5, 6], ['five', 'six'])))
print(dict(enumerate('SMTWTFS', 1)))
```

```
{5: 'five', 6: 'six'}
{1: 'S', 2: 'M', 3: 'T', 4: 'W', 5: 'T', 6: 'F', 7: 'S'}
```

generator

```
def gen():
    yield 1
    yield 2
    yield 'done'

g = gen()
print(g, type(g))
print(next(g))
print(*g)

for i in gen():
    print(i)
```

```
<generator object gen at 0x784e74d84a90> <class 'generator'>
1
2 done
1
2
done
```

2. fibonacci

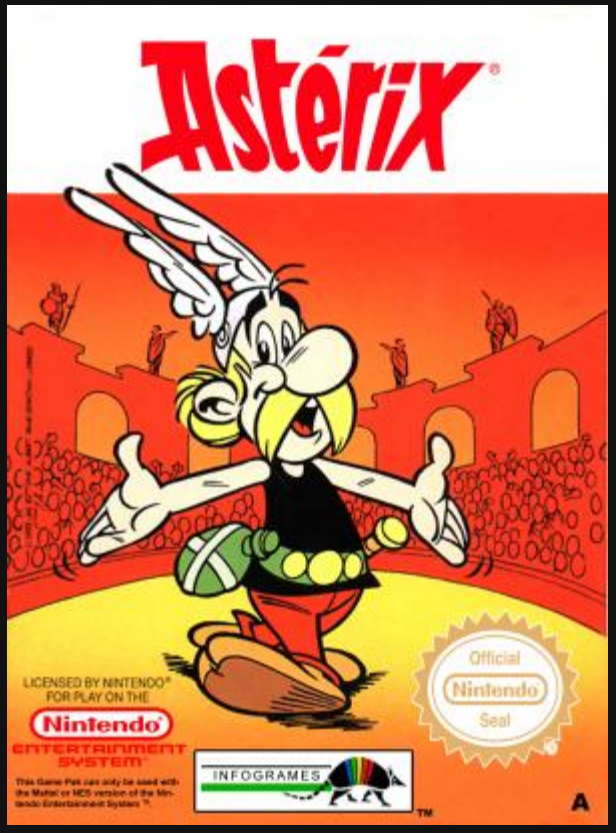
create the fibonacci series generator

```
0
1
```

summary

1. iterator can saves the states for future uses
2. custom iterator can be created using generator
3. generator is similar to function with **yield**

ASTERISKS


Astérix
LICENSED BY NINTENDO®
FOR PLAY ON THE
Nintendo
ENTERTAINMENT
SYSTEM
INFOGRAMES
Official
Nintendo
Seal
A

unpack

```
1 cpu, ram, (hdd, ssd) = 'Ryzen 7535U', [8, 8], (256, 512)
2 print(cpu, ram, hdd, ssd, sep='\n')
```

```
Ryzen 7535U
[8, 8]
256
512
```

```
a, b, c = '123'
print(a, b, c)
```

```
1 2 3
```

```
1 for i, c in enumerate('hi'):
2     print(i, c)
```

```
0 h
1 i
```

globbing

```
1 *x, y, z = range(3); print(x, y, z)
2 x, *y, z = range(3); print(x, y, z)
3 x, y, *z = range(3); print(x, y, z)
```

```
[0] 1 2
0 [1] 2
0 1 [2]
```

```
1 x, y, *z = (0, 1)
2 print(x, y, z)
3
4 x, y, *z = range(5)
5 print(x, y, z)
```

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

spread

```
1 print([1, 2, *[4, *[5, 6]]])
2 print(*[1, 2, *[4, *[5, 6]]])
```

```
[1, 2, 4, 5, 6]
1 2 4 5 6
```

```
a = [ 1, 2, 3 ]
b = [ 4, 5, 6 ]

x = [ a, b ]
print(x)

y = [ *a, *b ]  # concat values
print(y)

b.extend(a)
print(b)
```

```
[[1, 2, 3], [4, 5, 6]]
[1, 2, 3, 4, 5, 6]
[4, 5, 6, 1, 2, 3]
```

dict

```
1 x = {1:'a', 2:'b'}
2 y = x
3 y[1] = 'changed'
4 print(x)
5 print(y)
6 print(id(x), id(y))
```

```
{1: 'changed', 2: 'b'}
{1: 'changed', 2: 'b'}
137182534654976 137182534654976
```

```
1 x = {1:'a', 2:'b'}
2 y = {3:'c', 4:'d'}
3 z = x.copy()
4 z.update(y)
5 print(id(x), id(y), id(z))
6
7 print({**x, **y})
```

```
125009643133952 125009641105792 125009641104512
{1: 'a', 2: 'b', 3: 'c', 4: 'd'}
```

is and equals

```
a = list()
b = a
c = a.copy()
print(a is b)
print(a == b)

print(a is c)
print(a == c)
```

```
True
True
False
True
```

function params

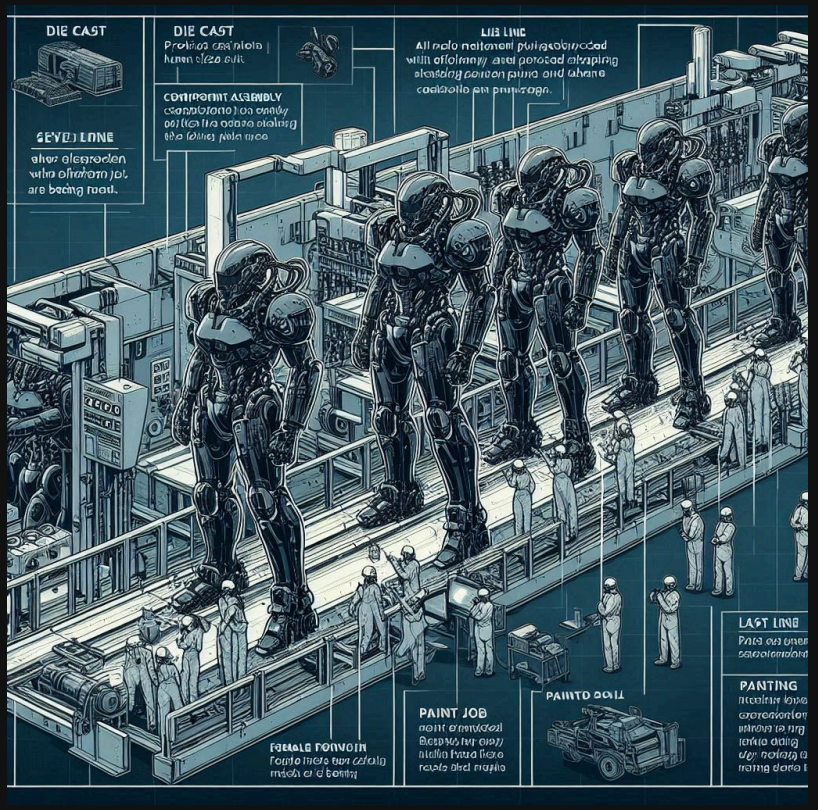
```
def foo(*args, **kwargs):
    print(args, kwargs)
    print()

foo(1, 2, 3)
foo(a=1, b=2, c=3)
a, d = [0, 1], {'a':3, 'b':4}
foo(a, d)
foo(*a, **d)
```

```
(1, 2, 3) {}

() {'a': 1, 'b': 2, 'c': 3}

([0, 1], {'a': 3, 'b': 4}) {}

(0, 1) {'a': 3, 'b': 4}
```

summary

1. help is writing clean ✨ and concise code
2. ordering matters in dictionary
3. can be used in function parameters

CLASSES



syntax

```
class ClassName:  # 🐫 camelCase naming convention
    pass

print(type(ClassName))   # types checks the object
instance = ClassName()   # creating the object
print(type(instance))
print(isinstance(instance, ClassName))
```

```
<class 'type'>
<class '__main__.ClassName'>
True
```

members

```
class Jar:
    cookie = 'choco'   # member variable
    chips  = 'potato'  # member variable

    def has():  # member funtion / method of Jar
        print('{0.cookie} cookies'.format(Jar))
        print('{0.chips} chips'.format(Jar))

print(Jar.cookie)
print(Jar.chips)
Jar.has()
```

```
choco
potato
choco cookies
potato chips
```

instance

```
class Jar:
    cookie = 'choco'   # member variable
    chips  = 'potato'  # member variable

    def has():  # member funtion / method of Jar
        print('{0.cookie} cookies'.format(Jar))
        print('{0.chips} chips'.format(Jar))

j = Jar()
print(j)
print(j.cookie)
try:
    j.has()
except TypeError as e:
    print(e)
```

```
<__main__.Jar object at 0x7cf315146f90>
choco
Jar.has() takes 0 positional arguments but 1 was given
```

self

```
class Jar:
    cookie = 'choco'   # member variable
    chips  = 'potato'  # member variable

    def has(self):  # member funtion / method of Jar
        print('{0.cookie} cookies'.format(Jar))
        print('{0.chips} chips'.format(Jar))

j = Jar()
print(j)
print(j.cookie)
j.has()
```

```
<__main__.Jar object at 0x7b43eaf470e0>
choco
choco cookies
potato chips
```

methods

```
class Car:
    bar = 'static member variable'
    def make_rar(self):  # 🐍 snake_case
        '''instance method has self as first argument'''
        self.rar = 'instance member variable'

print(Car.bar)
try:
    print(Car.rar)  # WHY?
except AttributeError as e:
    print(e)

c = Car(); print(c.bar)  # static
c.make_rar(); print(c.rar)  # instance
```

```
static member variable
type object 'Car' has no attribute 'rar'
static member variable
instance member variable
```

constructor

```
class Zoo:
    bar = 'static member variable'
    def __init__(self):
        '''Initializer Method aka Constructor'''
        self.rar = 'instance member variable'

    def show(self):
        print(self.rar)       # instance variable
        print(self.bar)       # static variable
        print(Zoo.bar)        # static variable

z = Zoo()
z.show()
```

```
instance member variable
static member variable
static member variable
```

arguments

```
class Car:
    def __init__(self, a, b=2):
        print(a, b)

Car(1)
Car(4, 5)
```

```
1 2
4 5
```

summary

1. class implement OOP concepts
2. blueprint for an object
3. object is the instance of the class
4. each object has its own data space

CALLABLE

function and class

```
def twice(x):
    return x * 2

class Square:
    def __init__(self, length):
        self.length = length

    def area(self):
        return self.length ** 2

print(type(twice), callable(twice), twice(4))
print(type(Square), callable(Square), s:=Square(4))
print(type(s.area), callable(s.area), s.area())
```

```
<class 'function'> True 8
<class 'type'> True <__main__.Square object at 0x76ef4effffb0>
<class 'method'> True 16
```

why?

```
def square(x):
    return x * x

print(callable(square))
print(square(5))

square = 5
square()
```

__name__

```
def square(x):
    return x ** 2

print(square.__name__)

foo = square
print(foo.__name__) # original name

print(id(foo), id(square))
del square  # remove source
print(foo(5))
```

```
square
square
133632463053632 133632463053632
25
```

passing function

```
def for_each(func, lst):
    output = []
    for i, e in enumerate(lst):
        output.append(func(e))
    return 'for_each({}):'.format(func.__name__), output
```

```
a = [0, 1, -2, 3.4]

print(*for_each(abs, a))
print(*for_each(int, a))
```

```
for_each(abs): [0, 1, 2, 3.4]
for_each(int): [0, 1, -2, 3]
```

map()

```
a = [0, 1, -2, 3.4]

print('map(abs, a):', list(map(abs, a)))
print('map(int, a):', list(map(int, a)))
```

```
map(abs, a): [0, 1, 2, 3.4]
map(int, a): [0, 1, -2, 3]
```

```
# passing custom function
def twice(x):
    return x * 2

print(*map(twice, range(10)))
```

```
0 2 4 6 8 10 12 14 16 18
```

λ function

```
def twice(x):
    return x * 2

print(twice(5))
print((lambda x: x*2)(5))
# IIFE ^^^ Immediately Invoked Function Expression
```

```
10
10
```

$$F = \frac{9}{5}C + 32$$

```
# Celsius and Fahrenheit conversion

print(f2c(c2f(36.5)))
```

```
36.5
```

wrap

```
print('pow(2, 3):', pow(2, 3))
try:
    print(*map(pow, range(10)))
except TypeError as e:
    print(e)
```

```
pow(2, 3): 8
pow() missing required argument 'exp' (pos 2)
```

```
print(list(map(lambda a: pow(a, 2), range(5))))
```

```
[0, 1, 4, 9, 16]
```

```
a = [0, 1, -2, 'three']

print(list(map(lambda a: str(a)[0], a)))  # first char
print(list(map(lambda a: str(a)[::-1], a)))  # reversed
```

```
['0', '1', '-', 't']
['0', '1', '2-', 'eerht']
```

1. top 5 most used words 

```
print(sorted([3, 1, 4, 2, 8]))
d = {'first' : 1, 'last': 26}
print(sorted(d))
print(sorted(d.items(), key=lambda k: k[1], reverse=True))
```

```
[1, 2, 3, 4, 8]
['first', 'last']
[('last', 26), ('first', 1)]
```

```
['the', 'and', 'of', 'I', 'to']
```

2. longest word possible

```
enter something: python
['python', 'typhoon']
longest word possible: typhoon
```

summary

1. function can be treated as variables
2. lambda is one liner function


EXCEPTION

runtime error

```
some_undefined_stuff  # NameError
1 + 'abc'             # TypeError
l = [0]; l[5]         # IndexError
d = {}; d['a']        # KeyError

2 / 0                 # ZeroDivisionError
```

try-catch

```
try:
    1/0
except:
    print('saved the world')
```

```
saved the world
```

traceback

```
import traceback

def a(x, y):
    def b():
        return x/y
    b()

try:
    a(1, 0)
except Exception:
    traceback.print_exc()
```

index

```
def get_one(iterator):
    try:
        return iterator[1]
    except (IndexError, KeyError) as e:
        print('An IndexError or KeyError occurred!')
        print(e)

print(get_one('exception'))
get_one({1: 'one'})
get_one([0, 1])

get_one({})
get_one('')
get_one([1])
```

```
x
An IndexError or KeyError occurred!
1
An IndexError or KeyError occurred!
string index out of range
An IndexError or KeyError occurred!
list index out of range
```

other-exceptions

```
raise Exception('description here')
raise RuntimeError('description here')
raise Warning('description here')
raise ValueError()
raise TypeError()
raise NameError('HiThere')
```

assert

```
import traceback

def divide_secure(number, divisor):
    assert divisor != 0, 'Divided a number by zero!'
    return number / divisor

print(divide_secure(10, 2))

try:
    print(divide_secure(10, 0))
except AssertionError as e:
    traceback.print_exc()
    print(e)
```

```
5.0
Traceback (most recent call last):
  File "<stdin>", line 11, in <module>
  File "<stdin>", line 5, in divide_secure
AssertionError: Divided a number by zero!
Divided a number by zero!
```

summary

1. **syntax errors**: caused by 😴 sleeping on the keyboard
2. **runtime errors**: error caused by sloppy programming
3. **semantic errors** error caused by missing class

COMPREHENSION

syntax

```
square = []
for i in range(10):
    square.append(i * i)

print(square)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

syntax

```
square = []
for i in range(10):
    square.append(i * i)

print(square)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
print([ i * i for i in range(10) ])
```

1. undumpify

```
d = iter('67726561742c20796f752063616e207265616421')

for i, j in zip(d, d):
    print(chr(int(i+j, 16)), end='')
```

filter

```
# square of even number till 10
square = []
for i in range(10):
    if i % 2 == 0:
        square.append(i * i)

print(square)
```

```
[0, 4, 16, 36, 64]
```

```
print(list(map(lambda i: i * i, (filter(lambda i: i % 2 == 0, ran
```

```
[0, 4, 16, 36, 64]
```

```
print([ i * i for i in range(10) if i % 2 == 0])
```

```
[0, 4, 16, 36, 64]
```

2. big-ones

How many 2-digit numbers have tens digit smaller than it unit digit i.e suppose for number 56, 5 is tens position, 6 is ones position 5 < 6

```
[12, 13, 14, 15, 16, 17, 18, 19, 23, 24, 25, 26, 27, 28, 29, 34, 35, 36, 3
36
```

2. big-ones

How many 2-digit numbers have tens digit smaller than it unit digit i.e suppose for number 56, 5 is tens position, 6 is ones position 5 < 6

```
numbers = []
for i in range(10, 100):
    tens, ones = divmod(i, 10)
    if tens < ones:
        numbers.append(i)

print(numbers)
print(len(numbers))
```

```
[12, 13, 14, 15, 16, 17, 18, 19, 23, 24, 25, 26, 27, 28, 29, 34, 35, 36, 3
36
```

3. number of digits

How many digits are there between 1-999?
5-15 : [5 6 7 8 9 10 11 12 13 14 15] = 17 digits

2889

reduce

```
import functools

print(sum(range(10)))
print(functools.reduce(lambda x, y: x + y, range(10)))
```

```
45
45
```

Q. find the factorial of 4 using functools.reduce()

dictionary

```
print({k : v for k, v in zip('hello', range(5)) if v % 2})
```

```
{'e': 1, 'l': 3}
```

PROJECT 03

write the program to break the ceasor cipher