verimli(fruitful)

```
def fun2(a,b):
    return a+b

print(fun2(5,6))
```

```
11
```

void(non-fruitful)

```
def fun1():
    print("Python")

fun1()
```

```
Python
```

```
a = 3
b = 5

multiply(3,5)

multiply(a,b)
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[7], line 4
      1 a = 3
      2 b = 5
----> 4 multiply(3,5)
      6 multiply(a,b)

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

The basic formula syntax of user-defined function is :

def function_name(parameters) : execution body

```
def multiply(a,b):
    print(a*b)

a = 3
b = 5

multiply(3,5)

multiply(a,b)
```

```
15
15
```

```
def motto() :
    print("Never give up!")

motto()
```

```
Never give up!
```

```
def multiply_1(a, b) :
    print(a * b)  # it prints something

def multiply_2(a, b) :
    return a * b  # returns any numeric data type value

multiply_1(10, 5)
multiply_2(10, 5)
```

```
50

50
```

```
print(type(multiply_1(10, 2)))
print(type(multiply_2(10, 5)))
```

```
20
<class 'NoneType'>
<class 'int'>
```

```
shadow_var = print("It can't be assigned to any variable")
print(shadow_var)  # NoneType value can't be used
```

```
It can't be assigned to any variable
None
```

```
def who(first, last) :  # 'first' and 'last' are the parameters(or
variables)
    print('Your first name is :', first)
    print('Your last name is :', last)

who('Guido', 'van Rossum')  # 'Guido' and 'van Rossum' are the
arguments
print()
who('Marry', 'Bold')  # 'Marry' and 'Bold' are also the arguments
```

```
Your first name is : Guido
Your last name is : van Rossum

Your first name is : Marry
Your last name is : Bold
```

```
who("Fatih")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 who("Fatih")

TypeError: who() missing 1 required positional argument: 'last'
```

Konumsal Argümanlar(Positional Args)

```
def pos_args(a, b):
    print(a, 'is the first argument')
    print(b, 'is the second argument')

pos_args(3,4)
print()
pos_args(4,3)
print()
pos_args("first","second")
print()
pos_args("second","first")
```

```
3 is the first argument
4 is the second argument

4 is the first argument
3 is the second argument

first is the first argument
second is the second argument

second is the first argument
first is the second argument
```

The formula syntax is : kwargs=values

```
def who(first, last) :  # 'first' and 'last' are the parameters(or variables)
    print('Your first name is :', first)
    print('Your last name is :', last)

who(last="van Rossum",first="Guido")
```

```
Your first name is : Guido
Your last name is : van Rossum
```

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
```

```
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")

parrot(1000) # 1 positional
parrot(voltage=1000) # 1 keyword args
parrot(voltage=10000, action="VOOOOM") # 2 keyword args
parrot("a million", "bereft of life", "jump") # 3 positional
parrot("1000", state="pushing up the daisies") # 1 positional, 1
keyword arg
```

```
-- This parrot wouldn't voom if you put 1000 volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's a stiff !
-- This parrot wouldn't voom if you put 1000 volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's a stiff !
-- This parrot wouldn't VOOOOM if you put 10000 volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's a stiff !
-- This parrot wouldn't jump if you put a million volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's bereft of life !
-- This parrot wouldn't voom if you put 1000 volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's pushing up the daisies !
```

```
parrot(voltage=5.0,"dead")
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Cell In[45], line 1
----> 1 parrot("dead",voltage=5.0 )

TypeError: parrot() got multiple values for argument 'voltage'
```

```
parrot("dead",voltage=5.0)
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Cell In[46], line 1
----> 1 parrot("dead",voltage=5.0 )

TypeError: parrot() got multiple values for argument 'voltage'
```

```
parrot(actor="jhonny Depp")
```

```
---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Cell In[47], line 1
----> 1 parrot(actor="jhonny Depp")

TypeError: parrot() got an unexpected keyword argument 'actor'
```

```
def function(a):
    pass  # actually, 'pass' does nothing. it just moves to the next
line of code

function(0, a=0)
```

```
---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Cell In[50], line 1
----> 1 function(0, a=0)

TypeError: function() got multiple values for argument 'a'
```

```
def a_function(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
```

```
def city(capital, continent='Europe'):
    print(capital, 'in', continent)

city("Athens")
city("Athens", continent="Greece")
city("Ulanbatoor", continent="Asia")
```

```
Athens in Europe
Athens in Greece
Ulanbatoor in Asia
```

1- A company is using the AWS Free Tier for several AWS services for an application.

What will happen if the Free Tier usage period expires or if the application use exceeds the Free Tier usage limits?

A. The company will be charged the standard pay-as-you-go service rates for the usage that exceeds the Free Tier usage.

B. AWS Support will contact the company to set up standard service charges.

C. The company will be charged for the services it consumed during the Free Tier period, plus additional charges for service consumption after the Free Tier period.

D. The company's AWS account will be frozen and can be restarted after a payment plan is established.

Answer:A

Which characteristics are advantages of using the AWS Cloud? (Choose two.)

A. A 100% service level agreement (SLA) for all AWS services

B. Compute capacity that is adjusted on demand

C. Availability of AWS Support for code development

D. Enhanced security

E. Increases in cost and complexity

Answer :B, D

A company wants to eliminate the need to guess infrastructure capacity before deployments. The company also wants to spend its budget on cloud resources only as the company uses the resources.

Which advantage of the AWS Cloud matches the company's requirements?

A. Reliability

B. Global reach

C. Economies of scale

D. Pay-as-you-go pricing

Answer: D

An online retail company has seasonal sales spikes several times a year, primarily around holidays. Demand is lower at other times. The company finds it difficult to predict the increasing infrastructure demand for each season.

Which advantages of moving to the AWS Cloud would MOST benefit the company? (Choose two.)

A. Global footprint

B. Elasticity

C. AWS service quotas

D. AWS shared responsibility model

E. Pay-as-you-go pricing

Answer: B, E

```
import math
math.pi
from math import pi
pi
```

```
from my_module import calculate_circle_area
print(calculate_circle_area(3))
```

```
28.274333882308138
```

-math.fabs(x) fonksiyonunu deneyelim (mutlak değeri döndürür) -math.floor(x) fonksiyonunu deneyelim (bu fonksiyon float değer alır ve x'ten küçük ya da eşit en büyük integer değeri döndürür)

```
import math
absolute = -5.25896
floor_test = 187.999

math.fabs(absolute)
math.floor(floor_test)
```

```
187
```

The formula syntax is : import module_name

import my_module # we've loaded my_module

my_module.my_function() # we've called a function defined in my_module

```
import my_module as mym # as kullanarak takma ad verdik
mym.calculate_circle_area(3)
```

```
28.274333882308138
```

```
import math
import my_module
import matplotlib_inline

import math as m
print(dir(math))
print(m.pi)
print(m.factorial(4))
print(m.log10(1000))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt',
'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e',
'erf', 'erfc', 'exp', 'exp2', 'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma',
'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm',
'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt',
'tan', 'tanh', 'tau', 'trunc', 'ulp']
3.141592653589793
24
3.0
```

```
import string as s
print(dir(s))
print(s.punctuation)
print(s.digits)
```

```
['Formatter', 'Template', '_ChainMap', '__all__', '__builtins__',
'__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', '_re', '_sentinel_dict', '_string',
'ascii_letters', 'ascii_lowercase', 'ascii_uppercase', 'capwords',
'digits', 'hexdigits', 'octdigits', 'printable', 'punctuation',
'whitespace']
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
0123456789
```

```
import datetime

print(datetime.datetime.now())
```

```
2023-08-08 14:46:53.443098
```

```
import random
city = ['Stockholm', 'Istanbul', 'Seul', 'Cape Town']
print(random.choices(city))
print(dir(random))
```

```
['Cape Town']
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random',
'SG_MAGICCONST', 'SystemRandom', 'TWOPI', '_ONE', '_Sequence', '_Set',
'__all__', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', '_accumulate',
'_acos', '_bisect', '_ceil', '_cos', '_e', '_exp', '_floor', '_index',
'_inst', '_isfinite', '_log', '_os', '_pi', '_random', '_repeat',
'_sha512', '_sin', '_sqrt', '_test', '_test_generator', '_urandom',
'_warn', 'betavariate', 'choice', 'choices', 'expovariate',
'gammavariate', 'gauss', 'getrandbits', 'getstate', 'lognormvariate',
'normalvariate', 'paretovariate', 'randbytes', 'randint', 'random',
'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular',
'uniform', 'vonmisesvariate', 'weibullvariate']
```

```
def getDoubleAlphabet(alphabet):
    doubleAlphabet = alphabet + alphabet
    return doubleAlphabet

def getMessage():
    stringToEncrypt = input("Please enter a message to encrypt: ")
    return stringToEncrypt

def getCipherKey():
    shiftAmount = input( "Please enter a key (whole number from 1-25):
")
    return shiftAmount
```

```
def encryptMessage(message, cipherKey, alphabet):
    encryptedMessage = ""
    uppercaseMessage = ""
    uppercaseMessage = message.upper()
    for currentCharacter in uppercaseMessage:
        position = alphabet.find(currentCharacter)
        newPosition = position + int(cipherKey)
        if currentCharacter in alphabet:
            encryptedMessage = encryptedMessage +
alphabet[newPosition]
        else:
            encryptedMessage = encryptedMessage + currentCharacter
    return encryptedMessage

def decryptMessage(message, cipherKey, alphabet):
    decryptKey = -1 * int(cipherKey)
    return encryptMessage(message, decryptKey, alphabet)

def runCaesarCipherProgram():
    myAlphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    print(f'Alphabet: {myAlphabet}')
    myAlphabet2 = getDoubleAlphabet(myAlphabet)
    print(f'Alphabet2: {myAlphabet2}')
    myMessage = getMessage()
    print(myMessage)
    myCipherKey = getCipherKey()
    print(myCipherKey)
    myEncryptedMessage = encryptMessage(myMessage, myCipherKey,
myAlphabet2)
    print(f'Encrypted Message: {myEncryptedMessage}')
    myDecryptedMessage = decryptMessage(myEncryptedMessage,
myCipherKey, myAlphabet2)
    print(f'Decypted Message: {myDecryptedMessage}')

runCaesarCipherProgram()
```

```
Alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Alphabet2: ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
bunu şifrelez
2
Encrypted Message: DWPW ŞKHTGNGB
Decypted Message: BUNU ŞIFRELEZ
```

```
f1 = open("diary.txt", "r", encoding="UTF-8") # UTF, “Unicode
Transformation Format”
print(f1.read())
f1.close()
```

```
f2 = open("diary2.txt", mode = "w")
f2.write("iyiyim sen nasilsin?")
f2.close()
```

```
merhaba günlük
```

open (file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

```
my_file = open("first_file.txt", mode="r")  # this syntax opens a
'txt' file
print(type(my_file))
my_file.close()
```

```
<class '_io.TextIOWrapper'>
```

MODES; Character What it's used for? 'r' Open for reading (default). If the file doesn't exist, FileNotFoundError will raise. 'a' Open for writing. It will append to the end of the file if it already exists. If there is no file, it will create it. 'w' Open for writing. It will be overwritten if the file already exists. If there is no file, it will create it. 'x' Open for exclusive creation, it will fail if the file already exists. 'b' Open in binary mode. 't' Open as a text file (default). '+' Open for updating (reading and writing).