COMPUTER SCIENCE

DISTRIBUTION OF MARKS

Unit No.	Unit Name	Marks
1	Computational Thinking and Programming — 2	40
2	Computer Networks	10
3	Database Management	20
	Total	**70**

12 GOLDEN RULES AND FACTS OF IDENTIFIER ARE AS FOLLOWING

1. It is name given to **program element**
2. Identifier name **cannot start with a digit**.
3. **Special Characters** are not allowed **(@,.,!,~,`,(,}.[,:,;,*,&,^,%,$,#)**
4. We can give any **valid name** to the identifier.
5. **Upper case** and **lower case letters** are **distinct**
6. Only **Alphabets, Digits and Underscores** are permitted.
7. An identifier is used for any variable, **function, data definition** etc.
8. Other **special characters are not allowed for naming a variable / identifier**
9. PYTHON is **case-sensitive** so that Uppercase Letters and Lower Case letters are different
10. The name of identifier **cannot begin with a digit.** However, Underscore can be used as first character while declaring the identifier.
11. Only alphabetic **characters, digits** and **underscore (_)** are permitted in PYTHON language for declaring identifier.
12. **Keywords cannot be used as Identifier.**

Here are some examples of valid and acceptable identifiers:

1.mohd	3.abc	5.a_123	7.e50	9.Ji	11. retVal
2.zara	4. move_na	6.mynam	8._temp	10. a23b9	12. abc_rr

Here are some examples of invalid and not acceptable identifiers:

1. sum*10	3.abc@	5.123a	7.e+50	9.10$*pay	11. Ret-Val
2.stu.name	4. move&name	6.Python!	8.12_temp	10. 123a23b9	12. pow(x,n)

IDENTIFY THE IDENTIFIER/ VARIABLE IN PYHTON USING 12 RULES

Example:	Ravi :	Valid , #x :	Invalid
1.N/10	17. e50isbus	33. Emp*Name	49. _temp
2.2j	18. INT	34. str	50. 23b9
3.A1b2c3	19. Stu*Name	35. a_b_c	51. Stu#Name
4.Age=	20. Nameofstudent	36. 3ab	52. Stu.Name
5.for	21. Ca_se	37. dobin123	53. int
6.class	22. xyz	38. len	54. Int
7.1abc	23. void	39. MoNo	55. _1
8.Stu&Name	24. _my	40. #ax	56. _
9.Name_of_student	25. Float	41. ax#	57. _ _A_ _b
10. Yk.sd	26. float	42. ax	58. _ _ _ _ _b
11. bool	27. student_name	43. Stubent_Name	59. true
12. Stu_Name	28. s.name	44. pow	60. True
13. a_b_c123	29. ifelse	45. move_na	61. false
14. 1Roll_NO	30. break	46. a_123	62. False
15. 1a2b3c	31. Stu_name	47. e50isbus	63. length
16. My.nameis.sanju	32. @abc	48. INT	64. Length

```
>>> a=10
>>> int=10
>>> a
10
>>> int
10
>>> a=int("10")
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    a=int("10")
TypeError: 'int' object is not callable
```

IMPORTANT POINT

FORMAT

```
x=17.12876
print("%f,%.3f ,%.4f,%.2f"%(x,x,x,x))
```

OUTPUT IS:

17.128760, 17.129, 17.1288, 17.13

OPERATORS

Arithmetic operators	→	+,-,*,**,/,//,%
Relational Operators	→	<,>,<=,>=,!=,==
Logical operators	→	and , or, not
Bitwise operators	→	&, |, ^ ,~ ,<< ,>>
Assignment operators	→	=
IDENTITY operators	→	is ,is not(ID)
Membership operators	→	In , not in
Type Casting	→	Date Type(value)

IF ELSE CONDITIONAL STATEMENTS

Ex. :

```
if x < 10:
  y = x
```

could be written:

```
if x < 10: y = x
```

but may not be written as

```
if x < 10:
  y = x
```

```
if x == 10:
    print('ten')
if 1:
    print('one')
```

always prints one, while the statement

```
if 0:
print('zero')
```

Syntax:

IF with OR and AND

```
x <= y and x <= z
x < y or x > y
if x == 1 or 2 or 3:
print("OK")
```

LOOPS -ITERATION STATEMENTS

```
for iterating_var in sequence:
    statements(s)
statements(s)
```

```
while expression:
    while expression:
        statement(s)
    statement(s)
```

```
Range(Begin, End-1 ,Step_Value)
For VN in List, Tuple, String :
    print (VN)
```

```
for i in range(6):
    print("*"*i)
```

```
range(start,end-1,step_value)

range(n)        #n>end
range(a,n)      #a>start,n>end
range(a,n,sv)   #a>start,n>end,sv>step value
```

end =n-1

Ex.

```
range(5)     ➡ 0,1,2,3,4
range(1,5)   ➡ 1,2,3,4
range(1,8,2) ➡ 1,3,5,7
range(0,7)   ➡ 0,1,2,3,4,5,6
```

STRING OUTPUTS

CHARACTER CONVERSION

```
n=ord("a")   #97
n=ord("A")   #65
ch=chr(65)   # A
```

STRING COMPARE FUNCTION'S

isalnum() | isalpha() | isdigit() islower() | isspace() | istitle() | isupper()

STRING CONVERSION FUNCTION'S

lower() | upper() | replace()

```
def fun(s):
    k=len(s)
    m=" "
    for i in range(o,k):
        if s[i].isupper():
            m=m+s[i].lower()
        elif s[i].isalpha():
            m=m+s[i].upper()
        else:
            m=m+'bb'
print(m)
fun('school2@com')
```

Ans. : SCHOOLbbbbCOM
(2 marks for correct output)

Note: Partial marking can also be given
(1/2 or 0.2 for each combination)

```
def fun(T):
    R=""
    l=len(T)
    for i in range(l):
        if T[i].isalpha()==False:
            R+='*'
        elif
    T[i].isupper()==True:
            R+=chr(ord(T[i])+1)
        else:
            R+= T[i+1]
    print("Final : ",R)
T= "Mind@work!"
print("Orignal : ",T)
```

Orignal : Mind@work!
Final : Nnd@*ork!*

Python Dictionary Cheatsheet

Cheat Sheet

CORE DEFINITION

Illustration

- Mutable collection of elements
- Key-Value Pairs enclosed in curly braces {}
- Unordered collection of items

KEY CHARACTERISTICS

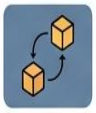
Illustration

- **Unique Keys**: Each key must be unique for its value
- **Immutable Keys**: Keys must be string, number, etc.
- **Separators**: Colon : between key/value; **Comma** , between pairs

CREATION METHODS

- **Empty Dict**: A = {} or A = dict()
- **With Data**: A = {1: "One", 2: "Two"}

ADDING & UPDATING



Illustration

- **Append:** A[new_key] = value → Adds new element
- **Immutable Ue:** A[existing_key] = value, number, etc.
- **With Data:** A = {1: "One", 2: "Two"}

ADDING & UPDATING

Caption

- **Append:** A[new_key] = value → Adds new element
- **Update Value:** A[existing_key] = new_value → Modifies value
- **update() Method:** A.update(B)

ADDING & UPDATING

Illustration

- **Append:** A[new_key] A.update(B)
  - Merges dictionary B into A
  - Overwrites values for existing keys → omnron B
  - Adds new keys from B

CREATION METHODS

Illustration

- **Empty Dict:** A = {} or A = dict()
- **With Data:** A = {1: "One", 2: "Two"}

Eximation

- **Append:** A[new_key] = value
- **Update Value:** A[existing_key] = new_value → Modifies value
- **update() Method:** A.update(B)
  - Merges dictionary B into A
  - Overwrites values for existing keys
  - Adds new keys from B

DELETION METHODS

Deleton

- **del statement**: del A[key]
  - Removes specific key-value pair
  - Raises KeyError if key is missing
- **pop() function**: A.pop(key)
  - Removes element by key

SYNTAX EXAMPLES

- A = {1: "One", 2: "Two"}
- A[3] = "Three" → {1: "One", 2: "Two", 3: "Three"}
- **del** A[1] → {2: "Two". 3: "Three"}

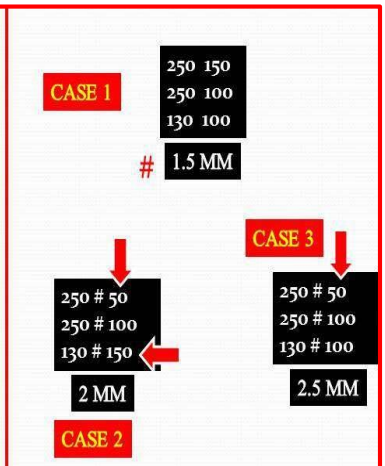
FUNCTION OUTPUT

```
def Change(P ,Q=30):
    P=P+Q
    Q=P-Q
    print( P,"#",Q)
    return (P)
R,S=150,100
R=Change(R,S)
print(R,"#",S)
S=Change(S)
```

Ans :

```
250 # 150   1
250 # 100   1
130 # 100   1
```

(1 MARK EACH FOR EACH CORRECT LINE)



RANDOM FUNCTION OUTPUT

```
import random
AR=[20,30,40,50,60,70];
FROM=random.randint(1,3)
TO=random.randint(2,4)
for K in range(FROM,TO+1):
    print (AR[K],end="#")
```

(i) 10#40#70# (ii) 30#40#50#
(iii) 50#60#70# (iv) 40#50#70#

Ans. : (ii) 30#40#50#
Maximum value FROM,TO is 3,4

(1/2 mark each for maximum value)
(1 mark for correct option)

randrange (start, stop, step)

Note: doesn't consider the last item i.e. it is exclusive. For example if **randrange (10,20,1)** will return any random number from **10 to 19** (exclusive).
it will **never select 20**.

randint randint(start, stop, step)

random.randint(0, 5) This will output either **1, 2, 3, 4 or 5.**

choice(list_name)

D= ['red', 'black', 'green']
random.choice(D) The **choice** function can often be used for choosing a random **element from a list**

shuffle(list_name)

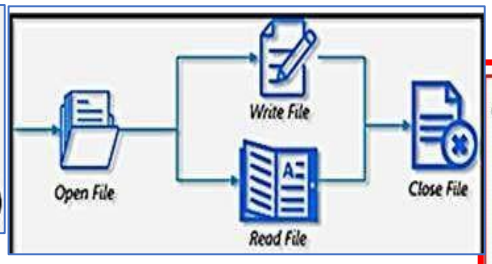
Note: shuffle function, shuffles the elements in list in place.

17

FILE HANDLING SYNTAX AND PROGRAMS

MODE
'r' : Read mode
'w' : Write mode
'a' : Appending mode
'r+' :Special read and write mode

```
open(FILE_NAME,MODE)
```




```
Obj_name.write("Data")
file = open("testfile.txt","w")
file.write("Hello World")
file.write("My Python File System...")
file.close()
```

```
Obj_name.read(bytes)
file = open("testfile.text", "r")
print file.read()  # file.read(5)
file = open("testfile.txt", "r")
print file.readline(): # file.readline(3)
```

OS Methods

	Syntax:
rename()	import os
remove()	os.rename(Old File, New File)
mkdir()	os.remove(File Name)
chdir()	os.mkdir("Folder Name")
getcwd()	os.chdir("Folder Name ")
rmdir()	os.getcwd()
	os.rmdir('dirname')
	os.rmdir("/tmp/test")

```
def COUNTLINES():
    file=open('STORY.TXT','r')
    lines = file.readlines()
    count=0
    for w in lines:
        if w[0]=="A" or w[0]=="a":
            count=count+1
    print("Total lines ",count)
    file.close()
```

(½ Mark for opening the file)
(½ Mark for reading all lines, and using loop)
(½ Mark for checking condition)
(½ Mark for printing lines)

Use of Split()

```
In file "Python is Fun"
fi=open("data.txt","r")
x=fi.readlines()
print(x)                #'Python is Fun'
print( x.split())       #['Python', 'is', 'Fun']
print( x.split("n"))    #['Pytho', ' is Fu', '']
```

Use of Join()

```
In file "PYTHON"
fi=open("data.txt","r")
x=fi.readlines()
print(x)                #PYTHON"
print("@".join(x) # 'P@Y@T@H@O@N'
```

Write a function in python to count the number of special symbols in each line text file 'data.txt' , the symbols like #,%,&,*,@ ,.,),],/,? any .

```
f1=open("data.txt","r")
lines=f1.readlines()
count=0
no_of_lines=len(lines)
print("Number of lines are : ",no_of_lines)
L=1
for line in lines:
    print("In line %d of"%L,end=" : ")
    for j in range(len(line)):
        if line[j].isalnum()==False:
            if line[j].isdigit()==False:
                print(line[j],end="")
                count=count+1
    L=L+1
    print("special symbols are : %d"%count)
    print()
    count=0
```

OUTPUT
Number of lines are : 3
In line 1 of : **& special symbols are : 6
In line 2 of : ^$# special symbols are : 4
In line 3 of : *()special symbols are : 3

STACK (PUSH)

```
def push(stack,x):
    stack.append(x)
def display(stack):
    if len(stack)<=0:
        print("Stack empty ")
    else:
        for i in stack:
            print(i,end=" ")
```

PUSH
Display

STACK (POP)

```
def pop(stack):
    n = len(stack)
    if(n<=0):
        print("Stack Empty")
    else:
        stack.pop()
```