Python Data Types

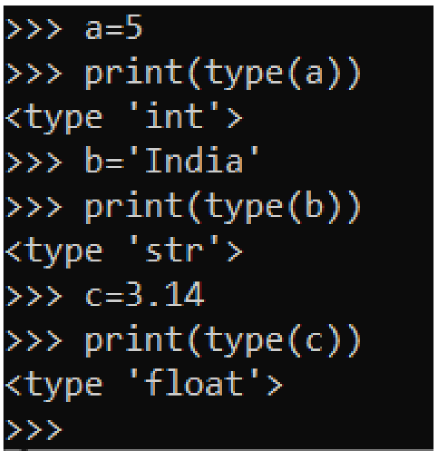
Built-in Data Types

- In programming, data type is an important concept.
- Variables can store data of different types, and different types can do different things.
- Python has the following data types built-in by default, in these categories:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

Getting the Data Type

- You can get the data type of any object by using the `type()` function:
- **Example:**



```
>>> a=5
>>> print(type(a))
<type 'int'>
>>> b='India'
>>> print(type(b))
<type 'str'>
>>> c=3.14
>>> print(type(c))
<type 'float'>
>>>
```

<u>Setting the Data Type</u>

- In Python, the data type is set when you assign a value to a variable:

Example	**Data Type**
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

<u>Setting the Specific Data Type</u>

- If you want to specify the data type, you can use the following constructor functions:

Example	**Data Type**
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Python Strings

- In Python, Strings are arrays of bytes representing Unicode characters.
- A string is a collection of one or more characters put in a single quote, double-quote or triple quote.
- In python there is no character data type, a character is a string of length one. It is represented by `str` class

String Literals

- String literals in python are surrounded by either single quotation marks, or double quotation marks.

Example:

`'hello'` is the same as `"hello"`.

You can display a string literal with the `print()` function:

Example

```
print("Hello")
print('Hello')
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"
print(a)
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

#You can use three double quotes:

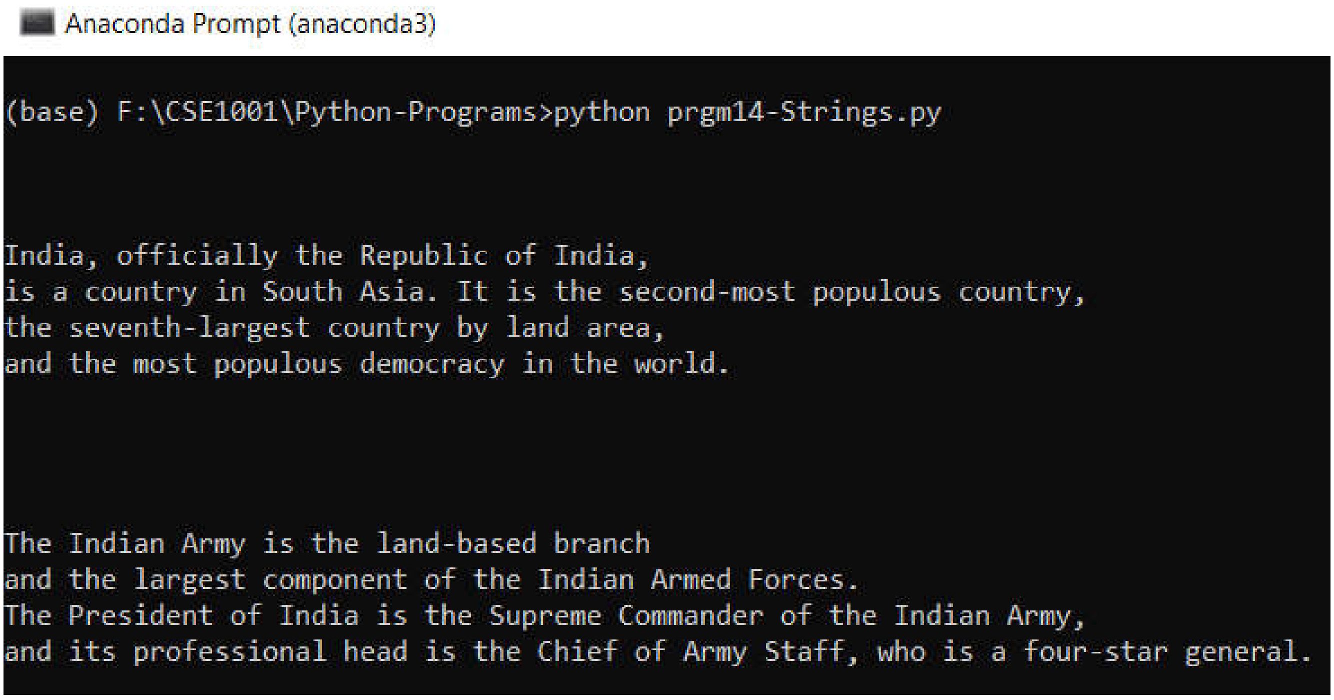
```
print("\n\n")

a = """India, officially the Republic of India,
is a country in South Asia. It is the second-most
populous country,
the seventh-largest country by land area,
and the most populous democracy in the world."""
print(a)
```

# three single quotes:

```
print("\n\n\n")

b = '''The Indian Army is the land-based branch
and the largest component of the Indian Armed Forces.
The President of India is the Supreme Commander of the
Indian Army,
and its professional head is the Chief of Army Staff,
who is a four-star general.'''
print(b)
```

Note: in the result, the line breaks are inserted at the same position as in the code.

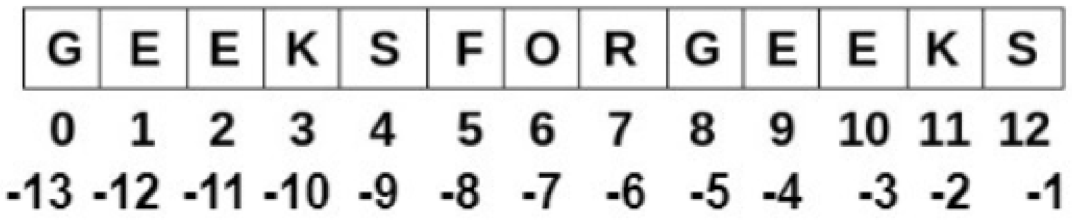
Output



```
(base) F:\CSE1001\Python-Programs>python prgm14-Strings.py



India, officially the Republic of India,
is a country in South Asia. It is the second-most populous country,
the seventh-largest country by land area,
and the most populous democracy in the world.





The Indian Army is the land-based branch
and the largest component of the Indian Armed Forces.
The President of India is the Supreme Commander of the Indian Army,
and its professional head is the Chief of Army Staff, who is a four-star general.
```

Accessing elements of String

- In Python, individual characters of a String can be accessed by using the method of Indexing.
- Square brackets can be used to access elements of the string.
- Indexing allows negative address references to access characters from the back of the String.
- Eg. -1 refers to the last character, -2 refers to the second last character and so on.



G	E	E	K	S	F	O	R	G	E	E	K	S
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Example:

```python
# Python Program to Access
# characters of String

String1 = "GeeksForGeeks"
print("Initial String: ")
print(String1)

# Printing First character
print("\nFirst character of String is: ")
print(String1[0])

# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])
```

Output:

```
Initial String:
GeeksForGeeks

First character of String is:
G

Last character of String is:
s
```

Example

- Get the character at position 1 (remember that the first character has the position 0):

```python
a = "Hello, World!"
print(a[1])
```

Slicing

- You can return a range of characters by using the slice syntax.
- Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

Negative Indexing

- Use negative indexes to start the slice from the end of the string:

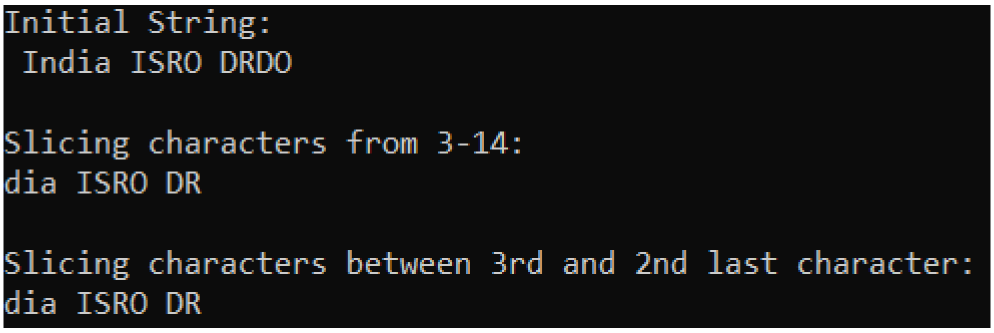
Example

Get the characters from position 3 to position 12 (not included), starting the count from the end of the string:

```
print("\n\n\n")

# Creating a String

String1 = " India ISRO DRDO"

print("Initial String: ")

print(String1)


# Printing 3rd to 14th character

print("\n Slicing characters from 3-14:")

print(String1[3:14])
```

```
# Printing characters between
# 3rd and 2nd last character
print("\nSlicing characters between " +
    "3rd and 2nd last character: ")
print(String1[3:-2])
```

Output:



```
Initial String:
 India ISRO DRDO

Slicing characters from 3-14:
dia ISRO DR

Slicing characters between 3rd and 2nd last character:
dia ISRO DR
```

String Length

To get the length of a string, use the `len()` function.

Example

The `len()` function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

String Methods

- Python has a set of built-in methods that you can use on strings.

Example

- The `strip()` method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

Example

The `lower()` method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

Example

The `upper()` method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

Example

The `replace()` method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Check String

- To check if a certain phrase or character is present in a string, we can use the keywords `in` or `not in`.

Example

Check if the phrase "Na" is present in the following text:

```
print("\n\n\n")

txt = "Nature is always beautiful"

x = "Na" in txt

print(x)
```

Example

Check if the phrase "ain" is NOT present in the following text:

```
txt = " Nature is always beautiful  "
x = "our" not in txt
print(x)
```

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable `a` with variable `b` into variable `c`:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

Example

To add a space between them, add a `" "`:

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

Output: Error

- But we can combine strings and numbers by using the `format()` method.

- The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:

Example

Use the `format()` method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

- The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {}
dollars."
print(myorder.format(quantity, itemno, price))
```

You can use index numbers `{0}` to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of
item {1}."
print(myorder.format(quantity, itemno, price))
```

Escape Character

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash `\` followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings" from the north."
```

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the
north."
```

String Methods

Python has a set of built-in methods that you can use on strings.

Note: All string methods returns new values. They do not change the original string.

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value

expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case

join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list

startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

How to Reverse a String in Python

- There is no built-in function to reverse a String in Python.
- Easy way is to use a slice that steps backwards, -1.

Example

Reverse the string "Hello World":

```
txt = "Hello World"[::-1]
print(txt)
```

Example:

We have a string, "Hello World", which we want to reverse:

The String to Reverse

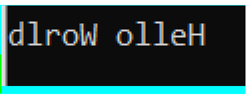
```
txt = "Hello World" [::-1]
print(txt)
```

Create a slice that starts at the end of the string, and moves backwards.

In this particular example, the slice statement `[::-1]` means start at the end of the string and end at position 0, move with the step `-1`, *negative* one, which means one step backwards.

Slice the String

```
print("\n\n\n")

txt= "Hello World"

txt1 = txt[::-1]

print(txt1)
```



```
dlroW olleH
```

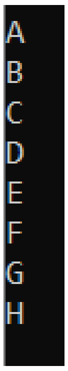
Now we have a string `txt` that reads "Hello World" backwards.

Processing strings using loops:

Example:

```
i=0

text="ABCDEFGH"

while i < len(text):

    print text[i]

    i += 1
```

Output:



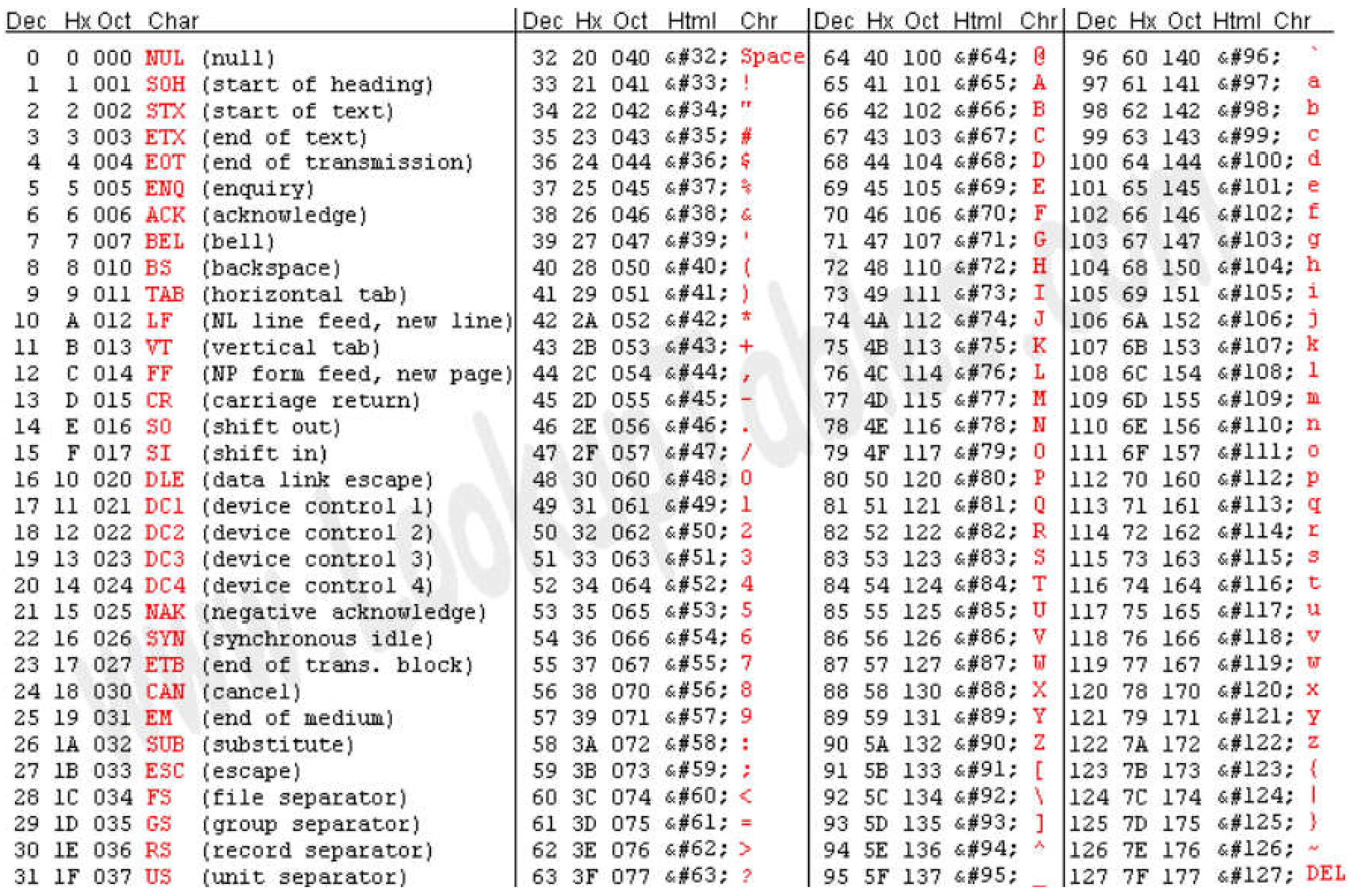
```
A
B
C
D
E
F
G
H
```

ASCII: American Standard Code for Information Interchange

- **American Standard Code for Information Interchange**, is a character encoding standard for electronic communication.

- ASCII codes represent text in computers, telecommunications equipment, and other devices.

- Most modern character-encoding schemes are based on ASCII.



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

#==

To get the ASCII code of a character, use the `ord()` function.

```
print(ord('h'))
104
Print(ord('a'))
97
```

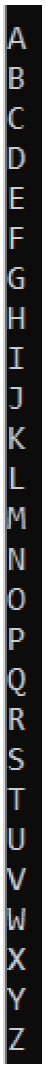
The `ord()` function returns the number representing the unicode code of a specified character.

===

Example:

```
print("\n\n\n")
i=65
j=1
while j <= 26:
    print (chr(i))
    i=i+1
    j=j+1
```

Output:


```
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
```

Activities using Python Coding:

1. Program to print all characters in a given string in forward and reverse order.

2. Program to count the number of vowels in a given string.

3. Program to check whether the given string is palindrome or not.

4. Program to check whether a given string (say string1) is present or not in another string (say string 2).
   EG:
   string 1: India
   String 2: India is our country.
   Output: String 'India' is present in string 'India is our Country'.

5. Program to print the following patterns.
   a.
   ```
   A
   B   B
   C   C   C
   D   D   D   D
   …
   ```
   b.
   ```
   A
   A   B
   A   B   C
   A   B   C   D
   ….
   ```