Section-3: Strings: Accessing characters and Substring in Strings, Data Encryption, Strings and Number Systems.

Accessing Characters and Substrings in Strings

Introduction:
A string in Python is an array of Unicode characters enclosed in quotes. Also, Python does not have a character data type; a single character is simply a string with a length of 1.
String indexing in Python is zero-based: the first character in the string has index 0 , the next has index 1, and so on.

Accessing Individual Characters:

In Python, we can access individual characters in a string using indexing. The characters in a string in Python can be accessed using both normal indexing and negative indexing.

- **Normal Indexing -** Each character in the string is assigned a numerical index starting from **0 to n-1**, where **n** is the length of the string. So characters in a string of size n, can be accessed from **0 to n-1**.

- **Negative Indexing -** A string will also have negative indexing. A negative index number starting from -1 is assigned from the last character in a string. So, **-1 for last character, -2 for 2nd from the last, -3 for 3rd from the last** and so on.

index	0	1	2	3	4	5	6	7
string	S	o	f	t	w	a	r	e
-ve Index	-8	-7	-6	-5	-4	-3	-2	-1

Individual characters in a string can be accessed by the **string name** followed by an **index number** in square brackets **[ ]**.

Syntax:

```
string_name [ index ]
```

Example:

```
# Accessing characters in strings
st = "Software Pros!"
print(st[0])    # Output: S
print(st[5])    # Output: a
print(st[0:4]) # Output: Soft
print(st[-1])  # Output: !
```

CT Students Only

Explanation:
In the above example, the first line creates a string variable st. The next three lines demonstrate how to access individual characters in the string using indexing.

- 1st print statement outputs the character at index 0, which is 'S'.
- 2nd print statement outputs the character at index 5, which is 'a'.
- 3rd print statement outputs the characters between 0 and 3, 'Soft'. (not including index 4).
- 4th print statement uses a negative index value to access the last character in the string, which is '!'.

Accessing Substrings:

In Python, you can **access substrings** from a string by **using slicing**. Slicing allows us to extract a portion of the original string by using the starting and ending index values.

String slicing is the process of obtaining a range of characters or a substring of a string by using its indices. Following are the 2 methods to slice a string.

1. **Array slicing (: operator)**
2. **slice() function**

1. **Array slicing (: operator)**

Definition:
Array slicing is used to obtain a portion of a string array or a list. It uses the **slicing operator :** and square brackets to slice a string.

Syntax:

object [start : stop : step]

start - start index of the slice (included),
stop - end index of the slice (excluded), and
step - step size is the number of elements to skip between each element in the slice

Application on Array Slicing	Application Find Palindrome

`s = "COLLEGE"`
`print(s[1:6])   # OLLEG    6 excluded`
`print(s[1:6:2]) # OLG      6 excluded`
`print(s[:3])    # COL      3 excluded`
`print(s[5:])    # GE       5 to last`
`# Negative index`
`print(s[-4:-1])      # LEG   -1 excluded`
`print(s[1:-4])       # OL    -4 excluded`
`print(s[5:1:-2]) #GL, in Reverse order`
`# Reverse`
`print(s[::-1])  # EGELLOC  String Reverse`
	`st1 = input("Enter a string : ")`
	`st2 = st1[:: - 1]`
	`if(st1 == st2):`
	`   print("This string is a Palindrome")`
	`else:`
	`   print("This string is not a Palindrome")`
	
	`Ex: level, madam, mom`

CIT Students Only

Table shows how the string sequence is sliced using **: operator**

Index	0	1	2	3	4	5	6
s	C	O	L	L	E	G	E
s[1:6]	C	O	L	L	E	G	E
s[1:6:2]	C	O	L	L	E	G	E
s[:3] s[0:end]	C	O	L	L	E	G	E
s[5:] s[beg :]	C	O	L	L	E	G	E

+ve index	0	1	2	3	4	5	6
-ve Index	-7	-6	-5	-4	-3	-2	-1
`s[-4:-1]`	C	O	L	L	E	G	E
`s[1:-4]`	C	O	L	L	E	G	E
`s[5:1:-2]`	C	O	L	L	E	G	E

`Reverse a string`							
`s[::-1]`	E	G	E	L	L	O	C

CIT Students Only

2. slice() Function

Definition:

1. The **slice() returns a slice object** (a portion size)
2. The **slice object is used as an index to slice** a sequence such as string, list, tuple, or range.

Syntax:

```
slice ( start , stop , step )
```

start - start index of the slice (included),
stop - end index of the slice (excluded), and
step - step size is the number of elements to skip between each element in the slice

Application: slice():

```
# slice() function
s = "Our CIT College!"
sub = slice(0, 3)    # Creates a slice object representing [0:3]
result = s[sub]      # Slices the string s using the slice object sub
print(result)       # Output: "Our"
```

Output: Our

String format methods

String formatting is the process of inserting a custom string or variable in predefined text. Python allows string formatting using one of the following 5 methods.

1. **% (String Format Operator)**
2. **format() method**
3. **f-strings**
4. **Built-in methods**
5. **String Template Class** (external module: **from string import Template**)

1. % (String Format Operator):

The % Operator is called a String Format Operator or an Interpolation Operator. It is used for simple positional formatting in strings. It allows you to insert values into a string, replacing placeholders with actual values. The placeholders are represented by percent signs followed by a format specifier that defines the type of the value being inserted.

Syntax:

```
<"format specifiers">  %  <data/vars>
```

- **format specifiers** - carries any string with %formatSpecifiers as placeholders (%d, %f, %s)
- ' **%** ' is the **String Format Operator** that substitutes data/variable value into format specifier
- **data/vars** - values to replace format specifiers

<"format specifiers"> may have format specifiers with Padding for data values as specified below:

%<fieldwidth>.<precision>f	**%6.2f**
%<fieldwidth>d	**%3d**
%<fieldwidth>s	**%10s**

<fieldwidth> is the total number of digits given for the value
<precision> is the number of decimal digits out of the given total digits
The unfilled digit positions will be added as padding spaces on the left.

CIT Students Only

Example:

```
name = "Raj"
age = 25
marks = 75.55
# without padding
print("Name:%s, Age:%d, Marks:%f" % (name, age, marks))
```

Output: Name:Raj, Age:25, Marks:75.550000

```
# with padding
print("Name:%10s, Age:%3d, Marks:%6.2f" % (name, age, marks))
```

Output: Name: Raj, Age: 25, Marks: 75.55

Table: List of format specifiers in Python

Format Specifier	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

CIT Students Only

2. format() method

The **format()** method formats the given values and insert them at placeholders in a string. The placeholders are represented by curly braces **{ }**.

Syntax1: `using format() with sequence of vars/values`

```
.format(var0,var1...)
print("{},{} ".format(var0,var1))
```

- format() method must be preceded by **.** operator
- var1, var2,...var-n are variables or values we pass into format() method
- placeholder { } is a value specifier.
  - Each pair of {}s represents a value from the variable passed into format()
  - The sequence of variables in the format() method must match the sequence of { } in quotes

Example:

```
name = "Venkat"
age = 20
print("My name is {} and I am {} years old.".format(name,age))
```

Output:

My name is Venkat and I am 20 years old.

Syntax2: `using format() with index number of vars/values`

```
.format(var0,var1...)
print("{var index0},{var index0},{var index1}".format(var0,var1))
```

- format() method must be preceded by **.** operator
- var1, var2,...var-n are variables or values we pass into format() method
- Each variable is indexed starting from 0 and increments by 1
- `{var index#}` represents the value of the variable specified in that position in the format(var0, var1, var2, ...) function.
- The index/position of variables in format() function starts with 0 and increments by 1

Example:

```
name = "Venkat"
age = 20
grade = 'A'
print("{0} has grade {2}. {0} is {1} years old.".format(name,age,grade))
```

Output:

Varsha has grade A. Venkat is 20 years old.

CR Students Only

3. f string format

a. **f** or **F** means formatted strings that are more readable and faster. (>= 3.6).
b. To create an f-string, prefix the string with letter "**f**".
c. These f strings contain replacement fields in curly braces **{ }**
d. The f or F in front of strings tells Python to look at the values, expressions, or objects inside { } and substitute them with the values of the given variables or expressions.
e. Formatted strings are evaluated at run time (while other string literals always have a constant value).

Example1: Basic fstrings

```
name1 = "Divya"
name2 = "Nitin"
cash1=5000
cash2=7000
total_cash = cash1 + cash2
#print in format method-2: Better one
print(f"Cash from {name1} = {cash1}")
print(f"Cash from {name2} = {cash2}")
print(f"Total amount = {total_cash}")
```

Output:
Cash from Nitin = 100
Cash from Naveen = 200
Total amount = 300

Example2: f string for precision, datetime and number conversion

```
import decimal
import datetime

# precision: nested fields, output: 12.35
width = 12
precision = 4
value = decimal.Decimal("12.3456789")
print(f"result:{value:{width}.{precision}}" )
print(f"result:{value:{2}.{5}}" )
# date format specifier, output: March 27, 2017
today = datetime.datetime(year=2023, month=3, day=17)
print(f"{today:%B %d, %Y}" )
```

CIT Students Only

```
# hex integer format specifier, output: 0x400
number = 1024
print(f"{number:#0x}" )
```

These are commonly used string format approaches in Python. We can customize the string format using different arguments and formatting options.

---

4. **Built-in methods to format strings**

In Python, the **class 'str'** provides several built-in methods to format or convert strings. The following table shows these methods and how they format the strings when they are used with a string object.

Table: Built-in methods to format strings

Method	Description	s="software Engineers"
s.capitalize()	converts the first character to uppercase.	Software Engineers
s.upper()	Converts all the characters in a string to uppercase.	SOFTWARE ENGINEERS
s.lower()	Converts all the characters in a string to lowercase.	software engineers
s.isupper()	Returns True if all the characters are uppercase. Otherwise, False	False
s.islower()	Returns True if all the characters are lowercase. Or else False.	False
s.find(substring, [start, end])	Returns the index of a specified character in the string or the start position of the given substring.	s.find("Eng") 9
s.count(substring,[start,end])	Counts the occurrence of a character or substring in a string.	s.count("r") 2
s.expandtabs([tabsize])	Replaces tabs defined by \t with spaces. Default tabsize = 8	
s.endswith(substring,[start, end])	Returns True if a string ends with the specified substring. False otherwise.	s.endswith("neers") True

CIT Students Only

s.startswith(substring, [start, end])	Returns True if a string starts with the specified substring. False otherwise.	s.startswith("Soft") True
s.isalnum()	Return True if all characters in a string are alphanumeric. False otherwise.	False
s.isalpha()	Return True if all characters in a string are alphabetic. False otherwise.	True
s.isdigit()	Return True if all characters in a string are digits. False otherwise.	False
s.split([separator],[maxsplit])	Splits a string separated by a separator(defaults is whitespace) and an optional **maxsplit** to determine the split limit. Returns a list.	["Software","Engineers"]
sep.join(sequence)	Takes all items in an iterable sequence (list, tuple, string), separates them by a given separator, and Joins them into a single string.	sep="_" seq="CIT" sep.join(seq) => C_I_T
s.replace(old, new,[maxreplace])	Replace old substring contained in the string s with a new substring.	s.("Engineers","Programmer") Software Programmers
s.swapcase()	Returns a new string with swapped case. i.e., uppercase becomes lowercase and vice versa.	sOFTWARE pROGRAMMERS
s.strip([characters])	Removes whitespaces or optional characters at the beginning and at the end of the string.	
s.lstrip([characters])	Removes leading whitespace or optional characters from a string.	
s.rstrip([characters])	Removes trailing spaces at the end of the string.	

CIT Students Only

Application: Using Built-in format methods

```
# built-in methods to format strings in class 'str'
s = "Software Pros"
print("capitalize:",s.capitalize() )
print("upper:",s.upper() )
print("lower:",s.lower() )
print("isupper:",s.isupper() )
print("islower:",s.islower() )
```

```
print("index# find:",s.find("Pros") )
print("count:",s.count("r") )
print("isnum:",s.isalnum() )
print("isalpha:",s.isalpha() )
print("isdigit:",s.isdigit() )
print("split:",s.split() )
print("join:",  "-".join(s) )
print("replace:",s.replace("Pros","Engineers"))
print("swapcase:",s.swapcase() )
```

Output:

capitalize: Software pros
upper: SOFTWARE PROS
lower: software pros
isupper: False
islower: False
index# find: 9
count: 2
isnum: False
isalpha: False
isdigit: False
split: ['Software', 'Pros']
join: S-o-f-t-w-a-r-e- -P-r-o-s
replace: Software Engineers
swapcase: sOFTWARE pROS

Operators for String Operations

Python provides the following operators for string operations:

- **String concatenation operator " + "**
- **String repetition operator " * "**
- String Slicing operator " : " to obtain substrings (See String slicing, p44)
- Indexing to traverse through strings (See Accessing Individual Character, p43)
- Membership operators (in, not in) to search for strings (See Operators in Unit-I)
- Relational operators (>, >=, <, <=) to compare strings (See Comparing Strings, p13)

Here, we will discuss + and * operators.
The + operator is used to concatenate 2 or more strings into one string.
The * operator is used to repeat a string up to a given number of times.

Operator	Purpose	Operation	Description
+	Concatenation	s1 + s2	Concatenates two strings, s1 and s2.
*	Repetition	s * n	Makes n copies of string s.

CRT Students Only

(+) Concatenation Operator:

Definition:

The + operator is used to join or concatenate two strings.
This concatenation operator in Python concatenates only objects of the same type.

Usage:

```
concatenate_string = string1 + string2   # concatenate the two strings
```

(*) Repetition Operator:

Definition:

The * operator is used to repeat a given string n number of times (similar to multiplication).

Usage:

```
repeat_string = string1 * n  # repeats string1 n times
```

Application:

```
# Concatenate & Repetition of strings
s1 = "Computer "
s2 = "Science"
s3 = s1 + s2
print(s3)

s4 = s1*3
print(s4)
```

Output:

Computer Science
Computer Computer Computer

String padding functions in Python

Definition:

In Python, String padding functions add extra characters such as spaces or zeros, at the start or end of a string to get a required length. Python does provide several built-in string padding functions for this purpose.

The commonly used string padding functions are,

1. **ljust(),**
2. **rjust(), and**
3. **center().**

Purpose:

These methods are very useful for formatting text in the form of tables or displaying information in a fixed-width format.

CIT Students Only

1. **ljust()**

Syntax:

```
svar.ljust(width[, fillchar])
```

ljust() function returns left-justified string of given width. The string is padded with **fillchar** (default is space) to make up the length.

Example:

```
s = 'Guntur'
padded_s = s.ljust(10, '*')
print(padded_s)    # Guntur****
```

2. **rjust()**

Syntax:

```
svar.rjust(width[, fillchar])
```

rjust() function returns right-justified string of given width. The string is padded with **fillchar** (default is space) to make up the length.

Example:

```
s = 'Guntur'
padded_s = s.rjust(10, '*')
print(padded_s)    # ****Guntur
```

3. **center()**

Syntax:

```
svar.center(width[, fillchar])
```

center() function returns centered string in the given width. The string is padded with **fillchar** (default is space) to make up the length.

Example:

```
s = 'Guntur'
padded_s = s.center(10, '*')
print(padded_s)    # **Guntur**
```

CIT Students Only

Data Encryption

Definition:

The process of converting information that cannot be understood by the unauthorized user is called data encryption. The reverse process is called decryption. Data encryption is used to protect the information transmitted over the network. The encrypted data prevents data corruption, sniffing, stealing, or security attacks.

The network protocols such as FTPS and HTTPS do provide security to the information transmitted over the network.

Security attacks:

Any action or a breach that compromises the security of information owned by an individual or an organization is called a security attack. Security attacks are classified into two: **Passive** and **Active**

- ➢ **Passive Attacks** - Unauthorized persons secretly reading or listening to private messages or message patterns while transmitting between a sender and a receiver.
- ➢ **Active Attacks** - Modification of the original data stream or the creation of a false data stream. Also includes,
    - Masquerade - one entity pretends to be a different entity
    - Replay- Passively capture and Unauthorized retransmission
    - **DOS (Denial Of Service)** - Disruption of an entire network

Process of Data Encryption:

- The information that is to be transmitted is called '**Plain Text**'.
- The **sender encrypts** the message by translating it into a secret code called '**Cipher Text**'.
- The **receiver decrypts** the cipher text into the original message or plain text.
- Both parties use **secret keys (public key & private key)** to encrypt and decrypt messages.
- **Caesar cipher** is a simple encryption method that has been in use for thousands of years.

Caesar cipher Encryption:

- Letter in a given plain text is changed to a letter that appears a certain number of positions farther down the alphabet set.
- For the characters near the end, the method goes back to the beginning of the alphabet set to locate the replacement characters.
- For example, if the distance value of a Caesar cipher is right-shift by 2 characters, the string "day" would be encrypted as "fca"

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
c	d	e	**f**	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	**a**	b

Of Students Only

Application: Caesar cipher encryption

```
# Caesar Cypher Encryption - Method1
msg = input('Enter your message: ')
dist= int(input('Enter cipher distance: '))
cmsg=""
for ch in msg:
    ordnum=ord(ch)
    ciphernum=ordnum+dist
    if ciphernum>ord('z'):
        ciphernum=ord('a')+dist-(ord('z')-ordnum+1)
    cmsg=cmsg+chr(ciphernum)
print(cmsg)
```

```
# Caesar Cypher Encryption - Method2
msg = input('Enter your message: ')
dist= int(input('Enter cipher distance: '))
cmsg=""
for ch in msg:
    # Add space for space
    if ch==" ":
        cmsg+=" "
    # uppercase encryption
    elif (ch.isupper()):
        cmsg+=chr((ord(ch)+dist-65)%26+65)
    # lowercase encryption
    else:
        cmsg+=chr((ord(ch)+dist-97)%26+ 97)
print(cmsg)
```

Output:
Enter your message: day
Enter cipher distance: 2
fca

CIT Students Only

Application: Caesar cipher decryption

```
# Caesar Cypher Decryption
code=input('Enter your text: ')
dist=int(input('Enter distance: '))
msg=""
for ch in code:
    ordnum=ord(ch)
    ciphernum=ordnum-dist
    if ciphernum<ord('a'):
        ciphernum=ord('z')-(dist-(ord('a')-ordnum+1))
    msg=msg+chr(ciphernum)
print(msg)
```

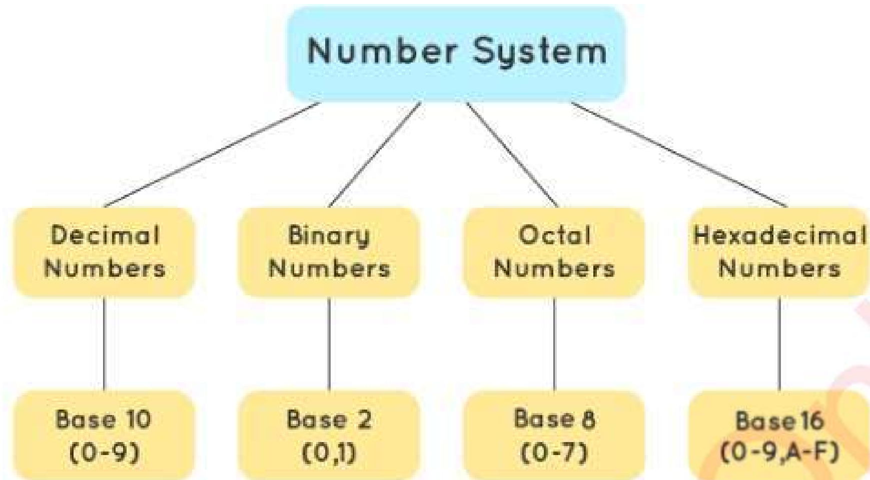
Output:
Enter your text: fca
Enter distance: 2
day

GIT Students Only

Number Systems

Number systems are the technique to represent numbers in the computer system architecture, every value that we save or read has a defined number system.
Computer architecture supports the following number systems.

1. **Binary number system**
2. **Octal number system**
3. **Decimal number system**
4. **Hexadecimal (hex) number system**



1) Binary Number System (Base: 2, Digits: 0, 1)
A Binary number system has only two digits 0 and 1. All binary numbers are represented in 0s and 1s.

2) Octal number system (Base: 8, Digits: 0-7)
Octal number system has only 8 digits from 0 to 7. All octal numbers are represented in 0,1,2,3,4,5,6 and 7.

3) Decimal number system (Base: 10, Digits: 0-9)
Decimal number system has only 10 digits from 0 to 9. All decimal numbers are represented in 0,1,2,3,4,5,6, 7,8, and 9.

4) Hexadecimal number system (Base: 16, Digits: 0-9, A-F)
A Hexadecimal number system has 16 alphanumeric values from 0 to 9 and A to F. All hexadecimal numbers are represented in 0,1,2,3,4,5,6, 7,8,9,A,B,C,D,E, and F. Here A is 10, B is 11, C is 12, D is 13, E is 14 and F is 15.

Table: Number Systems & Representation in Python

Number system	Base	Digits used	Example	Python assignment
Binary	**2**	**0,1**	**$(11110000)_2$**	**var = 0b11110000**
Octal	**8**	**0,1,2,3,4,5,6,7**	**$(360)_8$**	**var = 0o360**
Decimal	**10**	**0,1,2,3,4,5,6,7,8,9**	**$(240)_{10}$**	**var = 240**
Hexadecimal	**16**	**0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F**	**$(F0)_{16}$**	**var = 0xF0**

CIT Students Only

Decimal to Binary Conversion:

- **Manual conversion** - Decimal number is divided by 2 until we get 1 or 0 as the final remainder.
  $28_{10} = 11100_2$

Base target	Decimal	Remainder
2	28	**0**
2	14	**0**
2	7	**1**
2	3	**1**
2	**1**	

Decimal to Octal Conversion:

- **Manual conversion** - Decimal number is divided by 8 until we get **0 to 7** as the final remainder.
  $28_{10} = 34_8$

Base target	Decimal	Remainder
8	28	**4**
8	**3**	

Decimal to Hexadecimal Conversion:

- **Manual conversion** - Decimal number is divided by 16 until we get **0 to 15** as the final remainder.
  $28_{10} = 1C_{16}$

Base target	Decimal	Remainder
16	28	**12 = C**
16	**1**	

Automatic conversion: Decimal to Binary, Octal, Hexadecimal

From decimal to binary, octal or hexadecimal, use **bin( ), oct(), hex()** functions respectively.
From binary, octal or hexadecimal to decimal, use **int(other num, base)** function..

CH Students Only

Application:

```
# Aim: Program to convert Decimal to Binary, Octal and Hexadecimal
# Decimal to Binary, Octal, Hexadecimal
n = 28
bn = bin(n)
ot = oct(n)
hx = hex(n)
print("Decimal to Binary ", n, "=", bn)
print("Decimal to Octal ", n, "=", ot)
print("Decimal to Hexadecimal ", n, "=", hx)

#Binary to Decimal
print("Binary to Decimal = ",int(bn,2))
#Octal to Decimal
print("Octal to Decimal = ",int(ot,8))
#Hexadecimal to Decimal
print("Hexa to Decimal = ",int(hx,16))
```

Output:

Decimal to Binary 28 = **0b**11100
Decimal to Octal 28 = **0o**34
Decimal to Hexadecimal 28 = **0x**1c
Binary to Decimal = 28
Octal to Decimal = 28
Hexa to Decimal = 28

Binary to Decimal Conversion

Binary Number = **11100** $_2$

1	1	1	0	0
1×2^4	1×2^3	1×2^2	0×2^1	0×2^0
16	8	4	0	0

= 16 + 8 + 4 + 0 + 0

Decimal number = 28 $_{10}$

CIT Students Only

Octal to Decimal Conversion

Octal Number is : **34** $_8$

3	4
3x8^1	**4x8^0**
24	**4**

= 24 + 4

Decimal number = 28 $_{10}$

Hexadecimal to Decimal Conversion

Hexadecimal Number is : **1c** $_{16}$

1	c = 12
1x16^1	**12x16^0**
16	**12**

= 16 + 124 + 4

Decimal number = 28 $_{10}$

CIT Students Only