CLASS 2

STRINGS

Strings

- Letters, special characters, spaces, digits
- Enclose in quotation marks or single quotes

hi = “hello there”

or

hi = ‘hello there’

- Both syntaxes are valid in python.

Strings: Arithmetic operations

- String datatype supports “+” and “*” arithmetic operators
- **“+”** operator is used for concatenation

```
hi = "hello there"
name = "ana"
greet = hi + name
greeting = hi + "  " + name
```

```
print(hi)
print(name)
print(greet)
print(greeting)
```

Result:

```
hello there
ana
hello thereana
hello there ana
```

Strings: Arithmetic operations

- String datatype supports “+” and “*” arithmetic operators
- **“*”** operator is used for **repetition** of the string

```
name = ‘Sharad’

print(name)
print(name * 3)
print((name + ‘ ’)*3)
```

Result:

```
Sharad
SharadSharadSharad
Sharad Sharad Sharad
```

Examples: String arithmetic operators

aCubeIT
Institute of Artificial Intelligence

Solve

- Step 1: Take any string **input** into variable name **text** from **console.**
- Concatenate the variable text with the string **"happy"**
- **Print** the result string 2 times with space in between.

```
text = input('type anything: ')
print((text + "happy"+" ")*5)
```

Result:

type anything: we
wehappy wehappy

Strings: Relational Operator

- String comparison is in the lexicographic order.
- **Ordering of the works in the order of alphabet**
- Example: like in **Dictionary**
- Example: Type the following in python notebook

```
print("a" < "aa")
print("aa" < "ab")
```

Result:

True

True

Logical Operators on Boolean

Quick Question

- **What is a Boolean data type?**

  True and False

- **What is logical operator?**

  Logical AND,OR,NOT

Logical Operators on Boolean

- a and b are variable names (with Boolean values)
- `If a is True then` **`not a is?`**

```
a = True
print(not a)
```

Result: False

```
a = 5 > 1
print(not a)
```

 # Here 5>1 is a true statement
Result: False

Examples: Logical Operators on Boolean

- **`a and b`** # True if both are True

```
Example 1:((5 > 1) and (4 > 2))
Example 2:((5 > 1) and (4 <= 2))
```

- **`a or b`** # True if either or both are True

```
Example 1: ((5 > 1) or (4 <= 2))
Example 2: ((1 > 5) or (4 <= 2))
```

Result:
True
False

Result:
True
False

Strings: Length

- len() is a function used to retrieve the length of the string in the parentheses
- Example:
  - s = “abc” len(s)
  - Output = 3

Strings: Length-Examples

- Find the length of the following strings

1. "Amar123"
2. "My name is sharad"
3. "123678"

Strings: Indexing

- Square brackets used to perform indexing into a string to get the value at a certain index / position.
  - Example: s = “abc”
  - **s[0] # evaluates to “a”**
  - **s[1] # evaluates to “b”**
  - **s[2] # evaluates to “c”**
  - s[3] # trying to index out of bounds, error
  - **s[-1] # evaluates to “c”**
  - s[-2] # evaluates to “b”
  - **s[-3] # evaluates to “a”**

Strings: Slicing

- Can slice strings using [start:stop:step]
- If given two numbers, [start:stop], step = 1 by default.
  - Example : s[2:4]
- You can also omit numbers and leave just colons
  - s[::]

Strings: Slicing- Exercises

- Slice the following sub strings from

  s = “abcdefgh”

- ? “def”
- ? “df”
- ? “abcdefgh”
- ? “hgfedcba”

Slices:

s[3:6] = def

s[3:6:2] = df

s[: :] = abcdefg

s[::-1] = hgfedcba

- **Try using negative slicing**

Strings: Immutable

- Strings are immutable
- They cannot be modified
- **Try this**
  - **s = “hello”**
  - **s[0] = “y”**

CONTROL FLOW

Control Flow:

- **if** condition
- **while** loop
- **for** loop

- In order to write useful programs, we almost always **need the ability to check conditions and change the behavior of the program accordingly**.

IF CONDITION

Branching : ‘ *if* ‘ Condition

```
if <condition>:
   <expression>
   <expression>
   ............
```

```
if <condition>:
   <expression>
   <expression>
   .............
else:
   <expression>
   <expression>
   ...................
```

```
if <condition>:
   <expression>
   <expression>
   ..........
elif:
   <expression>
   <expression>
   ..............
else:
   <expression>
   <expression>
   ..................
```

- If value in the **<condition>** is true, evaluate **<expression>** in the block.

Branching : ‘ *if:* ‘ Condition

```
if <condition>:
   <expression>
   <expression>
   ............
```

- Print grade as “**A**” if marks are greater than 90.

```
In [1]: marks = int(input('Enter Marks: '))
        if marks  > 90:
            grade = 'A'
        print(grade)

        Enter Marks: 45

        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        <ipython-input-1-0646a645560d> in <module>
              2 if marks  > 90:
              3     grade = 'A'
        ----> 4 print(grade)

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

Branching : ‘ *if:* ‘ Condition

- Example 2:

```
In [4]: grade = "Not assigned"
        marks =  int(input("Enter Marks: "))
        if marks  > 90:
            grade =  "A"
        print(grade)
```

```
Enter Marks: 45
Not assigned
```

- **Good Programming Practice: Always initialize your variables.**

Branching : ‘ *if: else:*‘ Condition

```
if <condition>:
   <expression>
   <expression>
   ..............
else:
   <expression>
   <expression>
   ...................
```

```
In [6]: grade = 'Not assigned'
        marks =  int(input("Enter Marks: "))
        if marks  >= 45:
            grade =  'Pass'
        else:
            grade = 'Fail'
        print(grade)
```

```
Enter Marks: 50
Pass
```

Branching : ‘ *if: elif: else:* ‘ Condition

```
if <condition>:
   <expression>
   <expression>
   ..........
elif:
   <expression>
   <expression>
   ..............
else:
   <expression>
   <expression>
   ..................
```

```
In [7]: grade = "Not assigned"
        marks =  int(input("Enter Marks: "))
        if marks  > 90:
            grade =  "A"
        elif ((marks > 80) and (marks <=  90)):
            grade = "B"
        elif ((marks > 70) and (marks <=  80)):
            grade = "C"
        elif ((marks > 50) and (marks <=  70)):
            grade = "D"
        else:
            grade = "F"
        print(grade)

        Enter Marks: 67
        D
```

Control Statements:

aCubeIT
Institute of Artificial Intelligence

Good Programming Practice:

Always check the boundary conditions & default logic.

Indentation:

- Indentation is **very important** in Python

- Indentation is the way you denote blocks of code

Practice Exercises

1. Accept input from the user for the weight of her luggage. If the weight exceeds 50 pounds then print the following message: "There is a $25 charge for luggage that heavy."

Regardless of the weight print a thank you message as below: "Thank you for your business."

Check the **indentation difference** in both **Print statements**

```
In [8]: weight = float(input("How many pounds does your suitcase weigh? "))
        if weight > 50:

            print("There is a $25 charge for luggage that heavy.")

        print("Thank you for your business.")
```

```
How many pounds does your suitcase weigh? 55
There is a $25 charge for luggage that heavy.
Thank you for your business.
```