* Datatypes:

(i) * Integers:

→ int eg: whole no. : 3, 100, 10

(ii) * Floating point:

Type: float

→ e.g: No. with decimal point
2.3 4.6 100.0

(iii) * Strings:

Type: str

e.g: ordered sequences of characters.

→ denoted by " " , ' ' , ''' '''
"hello" "suma" "2000"

→ It can be a no. It also counts white space as character

~~It's an DS:~~

(iv) * List:

Type: list

→ Ordered sequences of objects

→ ~~Detoted~~ Denoted by [ ]

e.g: [10, "hello", 200.3]

v) * Dictionaries:

Type: dict

→ Unordered key : Value ~~key~~ pairs.

e.g: {"mykey": "value", "name": "Aks"}

→ Dehoted by { }

Immutabe objects: objects whose state can't be modified after it's created. Or.
we can't change an objects that's in sequence

(vi)* ~~Typles~~ Tuples:
Type: tup
→ Ordered immutable sequences of objects.
e.g: (10, "hello", 200.3)

(vii)* Sets:
Type: set
→ Unordered collection of unique object.
e.g: {"a", "b"}

(viii)* Booleans:
Type: bool
e.g: logical value indicating True or False

i.e. slice of string.

→ From lists to
→ Data structures are more specialized in basic datatype and data objects because they can hold data objects in some sort of sequence or in some sort of mapping.

→ Slicing: Allows to grab subsection of multiple characters.
Syntax: [start: stop: step]

Start → It's a numerical index for slice start
Stop → It's index you will go up to (but not include)
Step → It's the size of the "jump" you take.

Index: 0 1 2 3 4

rev. Index: 0 -4 -3 -2 -1

Page No.:

* String:

→ "hello"

→ "This is a string"

→ "I'm a geek"

→ print("hello world one")
o/p: hello world one

→ print('hello \n world')
o/p: hello
world

→ For checking length we use len fn.

→ len('hello')
o/p 5

→ len('D aon')
o/p 4

→ Index string:

eg: mystring = "hello world"

hello world
0 1 2 3 4 5 6 7 8 9
0 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1

(i) mystring
o/p: 'hello world'

(ii) mystring [0]
o/p: 'H'

(iii) mystring [-3]
o/p: 'r'

Two way grab characters
Index or reverse index.

→ slicing:
Complicated because grabing a subsection of string. Give more than one characters.

eg: mystring = 'abcdef'
o/p: 'abcdef'

(i) grabing from c to f

→ mystring [2:]
o/p: 'cdef'

→ mystring [:3]
o/p: 'abc'

→ mystring [2:5]
o/p: 'cde'

→ mystring [:]
o/p: 'abcdef'

→ mystring [::2]
o/p: 'ace'

→ mystring [::3]
o/p: 'ad'

→ mystring [::-1]
o/p: 'fedcba'

→ mystring [2:3:2]
start stop step
o/p: 'c'

→ Immutability:

e.g. name = "Sam"
name[0] = 'p' (error: string is immutable).
(i.e. we can't reassign in this way).

→ Concatenation:

merging two string together (copper e.g. for reassigning).

→ e.g. name = "Sam"
#name[0] = 'p'
making pam orig name ["Sam"] by grabbing 'am' from 'Sam'

slicing: last_letters = nam[1:]
" = 'am'

concatenate:
'p' + last_letters.
o/p: 'Pam'

→ x = 'Hello World'
x + "it's beauty"
'Hello world it's beauty'

→ letter = 'z'
letter * 4
O/p: 'zzzz'

(String multiplication)

→ x = "Hello World"
x.upper()
O/p: 'HELLO WORLD' (it doesn't change string)

for changing we should do
x = x.upper()

(Upper case)

→ (Split string)

x.split()
['Hello', 'World']

x.split('o')
['Hell', 'Wrld']

→ String interpellation:
Injecting variable into your string.
Eg: my_name = "Jose"
print("Hello" + my_name).
Methods:
(i) .format() method
(ii) f-strings (formatted string literals)

(i) .format() method:

Syntax: 'String here {} then also {}'.format('something', 'something 2')

Page No.:

Eg:

→ print('This is string{}'.format('Insert'))
O/p: This is string Insert

→ print('The {} {} {}'.format('fox', 'brown', 'quick'))
O/p: The fox brown quick

→ print('The {2} {1} {0}'.format('fox', 'brown', 'quick'))
O/p: The quick brown fox

→ print('The {0} {0} {0}'.format('fox', 'brown', 'quick'))
O/p: The fox fox fox

→ Assign keyword: print("The {q}, {b} {f}".format(f='fox', b='brown', q='quick'))
O/p: The quick brown fox

f-strings

→ name = "Jon"
print(f'Hello, his name is {name}')
O/p: Hello, his name is Jon

→ name = "Jon"
age = 3
print(f'{name} is {age} years old')
O/p: Sam is 3 years old.