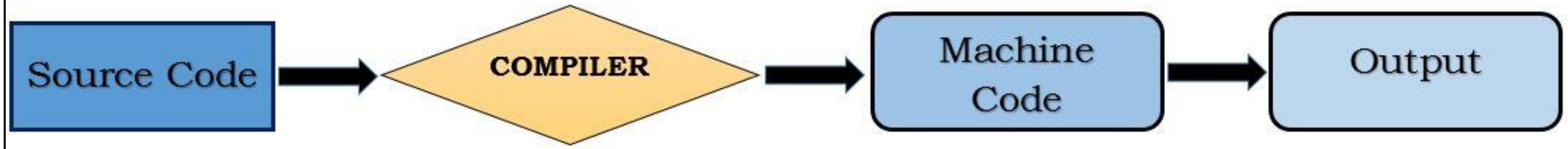
KLE Technological University
Creating Value
Leveraging Knowledge
B. V. Bhoomaraddi Engineering & Technology College Campus, Hubballi - India

Python Programming

Mr Praveenraj Pattar
Assistant Professor
School of Computer Science and Engineering
KLE Technological University.

COMPILER'S WORKING APPROACH



INTERPRETER'S WORKING APPROACH

Interpreter	Compiler
Definition: The definition of interpreter is a program that translates code written in high-level programming into machine code.	**Definition:** The simplest definition of a compiler is a program that translates code written in a high-level programming language (like C,C++ or Java) into low-level code (like Assembly) directly executable by the computer.
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyse the source code but the overall execution time is slower.	It takes large amount of time to analyse the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Basic Data types

- **Integers (int)**
- Real Numbers (float)
- Complex Numbers (complex)
- Strings (str)
- Boolean (bool)

Integers (int)

Examples:

- 15
- -25
- 98745632154698745216547896 32 –No range is defined, any value can be taken.
- 0o15(Octal):13
- 0XAB(Hexadecimal):171

Integers (int)

- Typecasting: float and string
  - int("25") : 25
  - int(231.65) : 231
- Conversion of different bases to integer type
- Syntax:

  int(value, base_value)

  1. int("25") : 25
  2. int("25",16) : 37
  3. int("25",8) : 21

Operations on Integers (int)

Expression	Result	Meaning
3 + 9	12	Addition
3 – 9	-6	Subtraction
2 ** 3	8	2 to the power of 3
2 * 3	6	Multiplication
10 / 4	2.5	Division
10 // 4	2	Integer Division
10 % 3	1	Modulus or Remainder
pow(2,3)	8	2 to the power of 3
abs(-105)	105	Absolute Value

Operations on Integers (int)…

Expression	Result	Meaning
bin(12)	0b1100	Converts integer to binary
oct(12)	0o14	Converts integer to octal
hex(12)	0xc	Converts integer to hexadecimal
2 & 3	2	Bitwise AND
2 | 3	3	Bitwise OR
2 ^ 3	1	Bitwise XOR
2 << 3	16	2 left-shifted 3 bits
15 >> 2	3	15 right-shifted 2 bits

Operations on Integers (int)…

Expression	Result	Meaning
2 < 3	True	2 is less than 3?
2 <= 3	True	2 is less than or equal to 3?
2 > 3	False	2 is greater than 3?
2 >= 3	False	2 is greater than or equal to 3?
2 == 3	False	2 is equal to 3?
2 != 3	True	2 is not equal to 3?

Basic Data types

- Integers (int)
- **Real Numbers (float)**
- Complex Numbers (complex)
- Strings (str)
- Boolean (bool)

Real Numbers (float)

Examples:

- 125.36
- 25.0
- 456.3e2
- 123.4567E3

Real Numbers (float)

Examples:

- Typecasting: float and string
  - float("25") : 25.0
  - float(231) : 231.0
  - float() : 0.0

Operations on Real Numbers (float)

Expression	Result	Meaning
1.2 + 0.1	1.3	Addition
3.2 – 9.1	-5.899999999999995	Subtraction
1.2 * * 0.1	1.0183993761470242090429 56	1.2 to the power of 0.1
2.5 * 3	7.5	Multiplication
7.5 / 3	2.5	Division
7.5 // 3	2.0	Integer Division
4.5 % 1.2	0.9000000000000001	Modulus or Remainder
pow(1.2,0.1)	1.0183993761470242090429 56	1.2 to the power of 0.1
abs(-1.2)	1.2	Absolute Value

Basic Data types

- Integers (int)
- Real Numbers (float)
- **Complex Numbers (complex)**
- Strings (str)
- Boolean (bool)

Complex Numbers (complex)

- These numbers are made of
  - Real Part
  - Imaginary Part
- Examples:
  - X + Yj or X + YJ
- Typecasting: string
  - complex("25+3j") : 25+3j
  - complex("231+1J") : 231+1J
  - complex("2") : 2+0j

Operations on Complex Numbers (complex)

Expression	Result	Meaning
(2+3j) + (5+6j)	(7+9j)	Addition
(2+3j) - (5+6j)	(-3-3j)	Subtraction
(2+3j) * (5+6j)	(-8+27j)	Multiplication
(2+3j) / (5+6j)	(0.4590163934426229+0.049180 32786885245j)	Division
(20+30j) // (2+3j)	Type Error	Integer Division
(12+31j) % (5+6j)	Type Error	Modulus or Remainder
1J ** 2	(-1+0j)	1J to the power of 2

Basic Data types

- Integers (int)
- Real Numbers (float)
- Complex Numbers (complex)
- **Strings (str)**
- Boolean (bool)

Strings(str)

- Strings can be enclosed in either single quotes (‘ ‘) or double quotes (“ “) or triple quotes (‘’’ ‘’’ or “”” “”””)
- Typecasting: int and float
  - str(“10”) : ‘10’
  - str (“10.25”): ’10.25’
- String Concatenation
  - “KLE” + “Tech” : 'KLETech'
  - X = ‘BVB’
  - Y = ‘CET’
  - X+Y : ‘BVBCET’

Operations on Strings(str)

Expression	Result	Meaning
'KLE'.lower()	'kle'	Converts all characters to lowercase.
'bvb'.upper()	'BVB'	Converts all characters to uppercase.
'kle TECHNOLOGICAL UNIVERSITY'.capitalize()	'Kle technological university'	Converts all characters to lowercase and first character to uppercase.
"Kle TEChnological univerSITY".swapcase()	'kLE tecHNOLOGICAL UNIVERsity'	Converts all lowercase characters to uppercase and vice-versa.
'bvb'.isalpha()	True	Returns true if all characters of given string are alphabetic and string is non-null, else false.
'Bvb'.isupper()	False	Returns true if all characters of given string are in uppercase and there is at least one alphabet, else false.

Basic Data types

- Integers (int)
- Real Numbers (float)
- Complex Numbers (complex)
- Strings (str)
- **Boolean (bool)**

Boolean (bool)

- Examples:
  - bool(0) : False
  - bool('') : False
  - bool(-12.5) : True
  - bool('KLE Tech') : True

Operations on Boolean(bool)

Expression	Result	Meaning
True and False	False	“AND” Operation
True or False	True	“OR” Operation
2 and 5	5	“AND” Operation
5 and 2	2	“AND” Operation
2 or 5	2	“OR” Operation
5 or 2	5	“OR” Operation
not False	True	“NOT” Operation
not True	False	“NOT” Operation
not 2	False	“NOT” Operation
not 0	True	“NOT” Operation

Identifiers

- Definition: It is name given to program element.
- Rules of framing identifier are as same in C, but also permits Unicode characters.
- Examples:

Valid	Invalid
KLE	1KLE
kle	k@le
_KLE	KLE#

Keywords

and	else	in	return
as	except	is	true
assert	false	lambda	try
break	finally	none	while
class	for	nonlocal	with
continue	from	not	yield
def	global	or	del
if	pass	elif	import
raise			

Variables

- Variables can be created when required and where required
  - x = 10 #x is created here of type int
  - x = 25 * x
  - y = x #y is created here of type int
- Variables value can be changed at any time
  - x = 10 #x is created here of type int
  - x = 25 * x #the value of x changes here
  - MAX =100   #MAX is created here
  - MAX = 200 #MAX value is changed
- The type of variable can be changed at any time
  - x = 2
  - type(x) :int
  - x = 2.5
  - type(x):float

Variables...

- The Operations permissible on variable depends on data type of variable.
  - x =25
  - x%7: 4
  - len(x): TypeError
  - x = “KLE”
  - x%7 : TypeError
  - len(x) : 3

Basic Input and Output

- Using print() function to print/output
  - print(“Hello”): Hello
  - print(12) : 12
  - print(12.3) : 12.3
  - print(“2.5”) : 2.5
  - print(1.5*3) : 4.5
  - print("Welcome","to","Learning","of","Python",'Programming',sep="----",end="\n*******")

    Welcome----to----Learning----of----Python----Programming

    ****
  -

Basic Input and Output...

- **format()** Function: It searches the strings for placeholders. These placeholders are indicated by braces ({ }) and indicate the some value needs to be substituted there.
- '{0} {1} {2} {3}'.format('Welcome','to','Python','Programming') :

  'Welcome to Python Programming‘
- '{2} {3} {0} {1}'.format('Welcomes','You','Python','Programming'):

  'Python Programming Welcomes You‘
- '{p} is {a}, {p} is {f}'.format\

(p='Python',a='easy',f='fun') : ‘Python is easy, Python is fun'

Basic Input and Output...

- Using input() Function to read
- Examples:
  - name = input("Enter your name:") : Enter your name:Praveenraj
  - lang = input("Enter the language:") : Enter the language:Python
  - print('{0} is learning {1}'.format(name,lang)) : Praveenraj is learning Python

Program Example

```
import math

x = int(input("Enter Number:"))

print('x = {}'.format(x))

print("square root (x) = {0} and square of x = {1}".format(math.sqrt(x),pow(x,2)))

print("The factorial of x = {}".format(math.factorial(x)))
```

Exercises

1) Write program to print trigonometry angles in radians from degrees.
2) Write a program to find the amount to be calculated after 5 years of period, with 2.5% as rate of interest and principal amount being ₹10,000.

Test Questions

1) Write a program to print the sum of square and cube of first N natural Numbers.
2) Write a program to find the surface area of right circular cone $\{\pi\, r\, (r + \sqrt{h^2 + r^2})\}$, base area of right circular cone $\{\pi\, r^2\}$ and volume of right circular cone$\{\pi\, r^2 (\frac{h}{3})\ \}$
3) Write a program to find the base area of cylinder $\{\pi\, r^2\}$, volume of cylinder$\{\pi\, r^2$ h $\}$ and surface area $\{2\, \pi\, r$ h + 2 $\pi\ r^2\}$

Control Statements

- Decision statements
  - if statement
  - if-else statement
  - if-elif-else statement
  - Nested if
- Loops
  - while
  - for

Control Statements

- Decision statements
  - **if statement**
  - if-else statement
  - if-elif-else statement
  - Nested if
- Loops
  - while
  - for

if statements

- Syntax :

```
if condition : statement

        or

if condition:

        statements

        ...
```

if statements

- Example 1

```
name = input("Enter the name:")
age  = int(input("Enter the age"))
if age>=18 : print("Hi {}, you can vote".format(name))
if age<18 : print("Hi {}, sorry you can't vote".format(name))
```

- Example 2

```
name = input("Enter the name:")
rank  = int(input("Enter the rank:"))
if rank>=6000 :
        print("Hi {}, you not eligible to take admission in KLE Tech".format(name))
if rank<6000 :
        print("Hi {}, you are eligible to take admission in KLE Tech".format(name))
```

Control Statements

- Decision statements
  - if statement
  - **if-else statement**
  - if-elif-else statement
  - Nested if
- Loops
  - while
  - for

if-else statements

- Syntax :

```
if condition : statement1
else         : statement2
```

OR

```
if condition :
        statements_block1
        ...
else :
        statements_block2
        ...
```

if-else statements...

- Example 1

```
name = input("Enter the name:")
age  = int(input("Enter the age"))
if age>=18 : print("Hi {}, you can vote".format(name))
else: print("Hi {}, sorry you can't vote".format(name))
```

- Example 2

```
name = input("Enter the name:")
rank  = int(input("Enter the rank:"))
if rank>=6000 :
        print("Hi {}, you not eligible to take admission in KLE Tech".format(name))
else :
        print("Hi {}, you are eligible to take admission in KLE Tech".format(name))
```

Control Statements

- Decision statements
  - if statement
  - if-else statement
  - **if-elif-else statement**
  - Nested if
- Loops
  - while
  - for

if-elif-else statements

- Syntax :

```
if condition :
        statements_block1
        ...
elif condition:
        statements_block2
        ...
elif condition :
        statements_block3
        ...

else :
        statements_blockN
```

Control Statements

- Decision statements
  - if statement
  - if-else statement
  - if-elif-else statement
  - **Nested if**
- Loops
  - while
  - for

Nested if statements

- Syntax :

```
if condition :
        statements_block1
        ...
        if condition:
                statements_block2
                ...
                if condition :
                        statements_block3
                        ...

else
        statements_blockN
```

Exercises

- A program to find the given point coordinates, in which of the quadrant of X-Y axis it lies.
- A program to find the roots of quadratic equations.

Control Statements

- Decision statements
  - if statement
  - if-else statement
  - if-elif-else statement
  - Nested if
- **Loops**
  - while
  - for

Loops

- Three rules to remember for loops
  1. Initialization
  2. Condition
  3. Update

Control Statements

- Decision statements
  - if statement
  - if-else statement
  - if-elif-else statement
  - Nested if
- Loops
  - **while**
  - for

While Loop

- syntax

```
while(condition):
        statements
        ...
```

While Loop...

- Example

```
number = int(input("Enter the Number"))
i=1
while i<=number:
        print(i)
        i = i+1
```

While Loop with else and break

- Example

```
number = int(input("Enter the positive integer:"))
i=2
while i<=number/2:
        if number%i==0:
                Prime = False
                break
        i = i+1
else:
    Prime = True
if Prime:
    print("Prime Number")
else:
    print("Composite")
```

Control Statements

- Decision statements
  - if statement
  - if-else statement
  - if-elif-else statement
  - Nested if
- Loops
  - while
  - **for**

for Loop

- syntax

```
for var in sequence: statement

        OR

for var in sequence:
        statements
        ...
```

for Loop...

- range() : Generates the value from 0 to n-1 value
- Types:

  range(end)

  range(start, end)

  range(start, end, step)

for Loop...

- Examples for different types:

```
for i in range(5):
    print(i)
for i in range(5, 10): #including start
    print(i)
for i in range(20, 10, -2): #including start
    print(i)
```

for Loop...

- Nested loops

```
for num in range(100,1000):
    sum = 0
    n = num
    while n>0:
        sum= sum+(n%10) ** 3
        n  = n // 10
    if num == sum:
        print("Armstrong Number:",num)
```

Lists

- Creating Lists
- Accessing List Elements
- Counting List Elements
- Searching List Elements
- List Slices
- Adding and Deleting Elements
- Adding, Multiplying and Copying Lists
- Nested Lists
- Operations on Lists

Lists

- **Creating Lists**
- Accessing List Elements
- Counting List Elements
- Searching List Elements
- List Slices
- Adding and Deleting Elements
- Adding, Multiplying and Copying Lists
- Nested Lists
- Operations on Lists

Lists

- List: It is ordered sequence of elements that can be dynamically altered.
- Syntax

```
list([ ])
```

- Example

```
L = list ([5,1,4,2,3])

    OR

L = [5,1,4,2,3]  #preferred
```

Lists...

- List: It is ordered sequence of elements that can be dynamically altered.
- Syntax

```
list([ ])
```

- Example

```
L = list ([5,1,4,2,3])

    OR

L = [5,1,4,2,3]  #preferred
```

Lists

- Creating Lists
- **Accessing List Elements**
- Counting List Elements
- Searching List Elements
- List Slices
- Adding and Deleting Elements
- Adding, Multiplying and Copying Lists
- Nested Lists
- Operations on Lists

Lists...

- Example

L = [5,1,4,2,3]

L[0]: 5

L[1]: 1

L[2]: 4

L[-1]: 3

L[-2]: 2

L[0] = 10

L : [10,1,4,2,3]

Lists

- Creating Lists
- Accessing List Elements
- **Counting List Elements**
- Searching List Elements
- List Slices
- Adding and Deleting Elements
- Adding, Multiplying and Copying Lists
- Nested Lists
- Operations on Lists

Lists...

- Example

  L = [5,1,4,2,3]

  len(L) : 5

Lists

- Creating Lists
- Accessing List Elements
- Counting List Elements
- **Searching List Elements**
- List Slices
- Adding and Deleting Elements
- Adding, Multiplying and Copying Lists
- Nested Lists
- Operations on Lists

Lists...

- Checking for Existence

```
>> L = [5,1,4,2,3]
>> 3 in L
True
>> 6 in L
False
```

- Counting Occurrences

```
>> L = [5,1,1,2,3]
>> L.count(1)
2
```

Lists...

- Locating Elements
  - list.index(x) #first occurrences of x value

```
>> L = [5,1,4,2,3]
>> L.index(2)
3
>> L.index(6)
ValueError
```

  - list.index(x,i) #search starts from index I instead of 0

```
>> L = [5,1,3,2,3]
>> L.index(3,3)
4
```

  - list.index(x,i,j) # search starts from index I but it will stop at index j

```
>> L = [5,1,3,2,3]
>> L.index(3,0,3)
2
```

Lists

- Creating Lists
- Accessing List Elements
- Counting List Elements
- Searching List Elements
- **List Slices**
- Adding and Deleting Elements
- Adding, Multiplying and Copying Lists
- Nested Lists
- Operations on Lists

Lists...

- List Slices : It is s sub-list extracted from original list.
- Syntax

  list[start:end]

- Examples

```
>> L = [5,1,4,2,3]
>> L[1:3]   #Slice of L from 1 to 3(excluding)
[1, 4]
>> L[1: ]   #Slice of L from 1 to end(excluding), here end is len(L)
[1, 4, 2, 3]
>> L[ : 3] #Slice of L from start to 3(excluding)
[ 5, 1, 4]
```

Lists...

- Examples

```
>> L = [5,1,4,2,3]
>> L[ : ]   #Slice entire list
[5,1,4,2,3]
>> L[1:3] = 3, 2
[5, 3, 2, 2, 3]
>> L[1:3] = 3, 0, 0, 2
[5, 3, 0, 0, 2, 2, 3]
```

Lists

- Creating Lists
- Accessing List Elements
- Counting List Elements
- Searching List Elements
- List Slices
- **Adding and Deleting Elements**
- Adding, Multiplying and Copying Lists
- Nested Lists
- Operations on Lists

Lists...

- Appending Elements
- list.append(x)
- Examples

```
>> L = [5,1,4,2,3]
>> L.append(9)
>> L
[5,1,4,2,3, 9]
>> L1 = [5,1,4,2,3]
>> L2 = [7, 8, 9, 6]
>> L1.extend(L2)
[5, 1, 4, 2, 3, 7, 8, 9, 6]
```

Lists...

- Inserting Elements
- list.insert(i, x) #Inserts x value at index i
- Examples

```
>> L = [5,1,4,2,3]
>> L.insert(2, 8)
>> L
[5,1,8,4,2,3, 9]
>> L1 = [5,1,4,2,3]
>> L1.insert(9,6)
>> L1
[5, 1, 4, 2, 3, 6]
```

Lists...

- Deleting Elements
- Using del
- Examples

```
>> L = [5, 1, 4, 2, 3]
>>  del L[1]
>> L
[5, 4, 2, 3]
>> L1 = [5,1,4,2,3]
>> del L1[1:3]
>> L1
[5, 2, 3]
```

Lists...

- Deleting Elements
- Using remove(x) #deletes a value of x
- Examples

```
>> L = [5, 1, 4, 2, 3]
>> L.remove(2)
>> L
[5, 1, 4, 3]
>> L1 = [5,1,4,2,3]
>> L1.remove(9)
>> L1
ValueError
```

Lists...

- Deleting Elements
- Using pop([i]) #deletes a index i
- Examples

```
>> L = [5, 1, 4, 2, 3]
>> L.pop(2)
4
>> L1 = [5,1,4,2,3]
>> L1.pop()
3
>> L1 = [5,1,4,2,3]
>> L1.pop(10)
IndexError
```

Lists...

- Deleting Elements
- Using clear() #deletes all
- Examples

```
>> L = [5, 1, 4, 2, 3]
>> L.clear()
>> L
[ ]
```

Lists

- Creating Lists
- Accessing List Elements
- Counting List Elements
- Searching List Elements
- List Slices
- Adding and Deleting Elements
- **Adding, Multiplying and Copying Lists**
- Nested Lists
- Operations on Lists

Lists...

- Adding Lists
- Examples

```
>> L1 = [5, 1, 4, 2, 3]
>> L2 = [3, 2, 1, 9, 6]
>> L3 = L1 + L2
>> L3
[5, 1, 4, 2, 3, 3, 2, 1, 9, 6]
```

- Multiplying List
- Examples

```
>> L1 = [5, 2, 3]  * 3
>> L1
 [5, 2, 3, 5, 2, 3, 5, 2, 3]
```

Lists

- Creating Lists
- Accessing List Elements
- Counting List Elements
- Searching List Elements
- List Slices
- Adding and Deleting Elements
- Adding, Multiplying and Copying Lists
- **Nested Lists**
- Operations on Lists

Lists...

- Adding Lists
- Examples

```
>> L1 = [5, 1, 4, 2, 3]
>> L2 = [3, 2, 1, 9, 6]
>> L3 = [1, L1, L2,556]
>> L3
[1, [5, 1, 4, 2, 3], [3, 2, 1, 9, 6], 556]
```

Lists

- Creating Lists
- Accessing List Elements
- Counting List Elements
- Searching List Elements
- List Slices
- Adding and Deleting Elements
- Adding, Multiplying and Copying Lists
- Nested Lists
- **Operations on Lists**

Lists...

- Examples

```
>> L1 = [5, 1, 4, 2, 3]
>> L2 = [3, 2, 1, 9, 6]
>> max(L2)
9
>> min(L1)
1
```

Tuples

- Creating Tuples
- Accessing Tuple Elements
- Counting Tuple Elements
- Searching Tuple Elements
- Tuple Slices
- Adding, Multiplying and Copying Tuples

Tuples

- **Creating Tuples**
- Accessing Tuple Elements
- Counting Tuple Elements
- Searching Tuple Elements
- Tuple Slices
- Adding, Multiplying and Copying Tuples

Tuples

- Tuple: It is immutable ordered sequence of elements.
- Syntax

  tuple([ ])

- Example

  T = tuple ([5,1,4,2,3])

  OR

  T = (5,1,4,2,3) #preferred

  OR

  T = 5,1,4,2,3

T = 5, #, indicating of tuple

Tuples

- Creating Tuples
- **Accessing Tuple Elements**
- **Counting Tuple Elements**
- **Searching Tuple Elements**
- **Tuple Slices**
- **Adding, Multiplying and Copying Tuples**

Sets

- Creating Sets
- Accessing Set Elements
- Counting Set Elements
- Searching Set Elements
- Adding and Deleting Elements
- Set Operations

Sets

- **Creating Sets**
- Accessing Set Elements
- Counting Set Elements
- Searching Set Elements
- Adding and Deleting Elements
- Set Operations

Sets

- Set: It is unordered collection of unique elements.
- Syntax

  set([ ])

- Example

  >> S = set ([5,1,4,2,3])

  >> S

  { 5, 1, 4, 2, 3}

  >> S = set(range(1,6))

  >> S

  { 1, 2, 3, 4, 5}

Sets

- Creating Sets
- **Accessing Set Elements**
- Counting Set Elements
- Searching Set Elements
- Adding and Deleting Elements
- Set Operations

Sets…

- Being unordered we cannot identify elements by indexing or by any other means. Therefore, the operations to check is for existences.

Sets

- Creating Sets
- Accessing Set Elements
- **Counting Set Elements**
- Searching Set Elements
- Adding and Deleting Elements
- Set Operations

Sets

- Example

```
>> S = set(range(1,6))
>> S
{ 1, 2, 3, 4, 5}
>>len(S)
5
```

Sets

- Creating Sets
- Accessing Set Elements
- Counting Set Elements
- **Searching Set Elements**
- Adding and Deleting Elements
- Set Operations

Sets

- Example

```
>> S = set(range(1,6))
>> 2 in S
True
>> 9 in S
False
```

Sets

- Creating Sets
- Accessing Set Elements
- Counting Set Elements
- Searching Set Elements
- **Adding and Deleting Elements**
- Set Operations

Sets

- Adding Individual Element
- Example

```
>> S = set(range(1,6))
>> S
{ 1, 2, 3, 4, 5}
>> S.add(9)
>> S
{ 1, 2, 3, 4, 5, 9}
```

Sets

- Adding Set of Elements
- Example

```
>> S1 = set(range(1,6))
>> S1
{ 1, 2, 3, 4, 5}
>> S2 = set(range(6,11))
>> S2
{ 6, 7, 8, 9, 10}
>> S1.update(S2)
>> S1
{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Sets

- Deleting Elements using pop() function would be deleted

#Since set is unordered we can't know which element

- Example

```
>> S1 = set(range(1,6))

>> S1

{ 1, 2, 3, 4, 5}

>> S1.pop()

1
```

Sets

- Deleting Elements using remove(x) function #X element would be removed
- Example

```
>> S1 = set(range(1,6))
>> S1
{ 1, 2, 3, 4, 5}
>> S1.remove(3)
>> S1
{ 1, 2, 4, 5}
```

Sets

- Deleting Elements using discard(x) function #X element would be removed
- Example

```
>> S1 = set(range(1,6))

>> S1

{ 1, 2, 3, 4, 5}

>> S1.remove(3)

>> S1

{ 1, 2, 4, 5}
```

Dictionaries

- Creating Sets
- Accessing Set Elements
- Counting Set Elements
- Searching Set Elements
- Adding and Deleting Elements
- Set Operations

Dictionaries

- **Creating dictionaries**
- Accessing dictionary Elements
- Counting dictionary Elements
- Searching dictionary Elements
- Adding and Deleting Elements

Dictionaries

- Dictionary: It is collection of key-value pairs subject to constrain that all the keys should be unique
- Syntax

  dict([ ])

- Example

```
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> D
{'apple': 'red', 'grapes': 'green'}
>> S = set(range(1,6))
>> S
{ 1, 2, 3, 4, 5}
```

Dictionaries

- Creating dictionaries
- **Accessing dictionary Elements**
- Counting dictionary Elements
- Searching dictionary Elements
- Adding and Deleting Elements

Dictionaries

- Example

```
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> D['apple']
'red'
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> D['Mango']
KeyError
```

Dictionaries

- Example

```
>> D = dict ([(‘apple’, ‘red’), (‘grapes’, ‘green’)])
>> D[‘grapes’] = ‘purple’
>> D
{'apple': 'red', 'grapes': ‘purple'}
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> D['mango']='yellow'
>> D
{'apple': 'red', 'grapes': 'green', 'mango': 'yellow'}
```

Dictionaries

- Creating dictionaries
- Accessing dictionary Elements
- **Counting dictionary Elements**
- Searching dictionary Elements
- Adding and Deleting Elements

Dictionaries

- Example

```
>> D = dict ([('apple', 'red'), ('grapes', 'green')])

>> len(D)

2

>> D = dict ([('apple', 'red'), ('grapes', 'green')])

>> for k in D: print(k)
apple
grapes
```

Dictionaries

- Example

```
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> k = D.keys()
>> for i in k: print(i)
apple
grapes
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> V = D.values()
>> for i in V: print(i)
red
green
```

Dictionaries

- Example

```
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> for K in D: print(D[K])
red
green
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> for K,V in D.items(): print(K,V)
apple red
grapes green
```

Dictionaries

- Creating dictionaries
- Accessing dictionary Elements
- Counting dictionary Elements
- **Searching dictionary Elements**
- Adding and Deleting Elements

Dictionaries

- Example

```
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> ‘apple’ in D
True
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> ‘mango’ in D
False
>> D = dict ([('apple', 'red'), ('grapes', 'green')])
>> ‘mango’  not in D
True
```

Strings

- Conversion(typecasting) From and To Strings
- Searching in Strings
- Splitting in Strings
- Joining Strings
- Modifying Strings
- Padding Strings

Strings

- **Conversion(typecasting) From and To Strings**
- Searching in Strings
- Splitting in Strings
- Joining Strings
- Modifying Strings
- Padding Strings

Strings(str)

- String: It is an immutable sequence of characters.
- Strings can be enclosed in either single quotes (‘ ‘) or double quotes (“ “) or triple quotes (‘’’ ‘’’ or “”” “””)
- Typecasting: int and float
  - str(“10”) : ‘10’
  - str (“10.25”): ’10.25’
- String Concatenation
  - “KLE” + “Tech” : 'KLETech'
  - X = ‘BVB’
  - Y = ‘CET’
  - X+Y : ‘BVBCET’

Strings

- Conversion(typecasting) From and To Strings
- **Searching in Strings**
- Splitting in Strings
- Joining Strings
- Modifying Strings
- Padding Strings

Strings

- Existence of substring using startswith() and endswith()
- Used to check if a string contains a particular substring at a particular position
- startswith() Function
- syntax

```
str.startswith(substring)

str.startswith(substring,start)

str.startswith(substring,start,end)
```

Strings

- Example for str.startswith(substring)

  >> “KLE Technological University”.startswith(“KL”)

  True

  >> "KLE Technological University".startswith("kle")

  False

- Example for str.startswith(substring,start)

  >> "KLE Technological University".startswith("Techn",4)

  True

  >> "KLE Technological University".startswith("Techn",1)

  False

Strings

- Example for str.startswith(substring,start,end)

```
>> "KLE Technological University".startswith("Techn",4, 10)
True
>> "KLE Technological University".startswith("Techn",4, 6)
False
```

Strings

- endswith() Function
- syntax

```
str.endswith(substring)

str.endswith(substring,start)

str.endswith(substring,start,end)
```

Strings

- Example for str.endswith(substring)

  >> "KLE Technological University".endswith("University")

  True

  >> "KLE Technological University".endswith("Univers")

  False

- Example for str.endswith(substring,start)

  >> "KLE Technological University".endswith("University",18)

  True

  >> "KLE Technological University".endswith("University",19)

  False

Strings

- Example for str.endswith(substring,start,end)

```
>> "KLE Technological University".endswith("University",18,28)
True
>> "KLE Technological University".endswith("University",18,25)
False
```

Strings

- Existence of substring using find(), rfind(), index() and rindex()
- find() Function: It searches for first occurrence of substring and returns index if found else -1

  str.find(substring)

  str.find(substring,start)

  str.find(substring,start,end)

>> "This is a demo".find("is") : 2

>> "This is a demo".find("is“, 3) : 5

>> "This is a demo".find("is",7) : -1

>> "This is a demo".find("is",3,15) : 5

>>"This is a demo".find("is",3,4) : -1

Strings

- rfind() Function: It searches for last(rightmost) occurrence of substring and returns index if found else -1

str.rfind(substring)

str.rfind(substring,start)

str.rfind(substring,start,end)

>> "This is a demo".rfind("is") : 5

>> "This is a demo".rfind("is“, 3) : 5

>> "This is a demo".rfind("is",7) : -1

>> "This is a demo".rfind("is",3,15) : 5

>>"This is a demo".rfind("is",3,5) : -1

Strings

- index() Function: It is similar to find(), but generates ValueError instead of -1
- rindex() Function : It is similar to rfind(), but generates ValueError instead of -1

Strings

- Conversion(typecasting) From and To Strings
- Searching in Strings
- **Splitting in Strings**
- Joining Strings
- Modifying Strings
- Padding Strings

Strings

- Splitting Strings Using partition() and rpartition()
- partition() and rpartition() allow extraction of part before separator, separator itself and after separator in string.

```
>> "India:New Delhi".partition(":")

('India', ':', 'New Delhi')

>>"India:New Delhi China:Beijing".rpartition(":")

('India:New Delhi China', ':', 'Beijing')
```

Strings

- Splitting Strings Using split() and rsplit()
- split() and rsplit() works for multiple delimiters.

```
>> "India:New Delhi China:Beijing".split(":")
['India', 'New Delhi China', 'Beijing']
>>"India:New Delhi China:Beijing".split()
['India:New', 'Delhi', 'China:Beijing']
```

Strings

- Conversion(typecasting) From and To Strings
- Searching in Strings
- Splitting in Strings
- **Joining Strings**
- Modifying Strings
- Padding Strings

Strings

- Example

```
>> " and ".join(("35","66","45"))
'35 and 66 and 45‘
>> " , ".join(("35","66","45"))
'35 , 66 , 45‘
>> " #* ".join(("35","66","45"))
'35 #* 66 #* 45'
```

Strings

- Conversion(typecasting) From and To Strings
- Searching in Strings
- Splitting in Strings
- Joining Strings
- **Modifying Strings**
- Padding Strings

Strings

- Splitting Strings Using lstrip(), rstrip and strip()
- lstrip() Function: It is left-strips the string by removing white spaces

>>"\n\n\n\t KLE Technological University".lstrip()

KLE Technological University

- rstrip() Functiom: It is right-strips the string by removing white spaces

>> "\n\n\n\t KLE Technological University \t \t\n\n".rstrip()

'\n\n\n\t KLE Technological University‘

- strip() Function: It left-strips and right-strips the string by removing white spaces

>> "\n\n\n\t KLE Technological University \t \t\n\n".strip()

'KLE Technological University'

Strings

- Substring Using replace()
- Syntax

str.replace(old,new)

- Example

>> "Delhi-Mumbai-Bengaluru-Mysore-Hubli-Dharwad".replace("Delhi","Vijaypura")

'Vijaypura-Mumbai-Bengaluru-Mysore-Hubli-Dharwad‘

>> 'Vijaypura-Mumbai-Bengaluru-Mysore-Hubli Dharwad'.replace("Dharwad","Hospete")

'Vijaypura-Mumbai-Bengaluru-Mysore-Hubli Hospete'

Strings

- Conversion(typecasting) From and To Strings
- Searching in Strings
- Splitting in Strings
- Joining Strings
- Modifying Strings
- **Padding Strings**

Strings

- ljust() Function : To left justify the content within expanded string
- rjust() Function : To right justify the content within expanded string
- center() Function : To center justify the content within expanded string
- Syntax

str.ljust(width)

```
>> "To left justify the content".ljust(50)
'To left justify the content                ‘
>> "To left justify the content".ljust(50,"*")
'To left justify the content**********************'
```

Strings

- Syntax

```
        str.rjust(width)
>> "To left justify the content".rjust(50)
'                To left justify the content‘
>> "To left justify the content".rjust(50,"*")
'**********************To left justify the content'
```

Strings

- Syntax

str.center(width)

```
>> "To left justify the content".center(50)
'                To left justify the content                 '
>> "To left justify the content".center(50,"*")
'**********To left justify the content**********'
```

Functions

1. Function Definition	8. Returning Single value from functions
2. Function Call	9. Returning Collections from functions
3. Positional Arguments	10. Global Variable
4. Default Arguments	11. Nested Functions
5. Keyword Arguments	12. Lambda Expressions
6. Variable Arguments	13. Unpacking Argument Lists
7. Returning from functions	

Functions

- Function Definition: It contains the body of the function, where instructions are written.
- Syntax

```
def function_name(parameters):
    function_body
    ...
```

```
>> def my_function():
    print("It my function definition!!!")
```

Functions

- Function Call: It is statement where programs get interrupted to execute different set of instructions.
- Syntax

```
function_name(parameters)
```

```
>> my_function()
```

```
It my function definition!!!
```

Functions

Positional Arguments

- It can receive arguments(parameters) from calling function and it can return arguments to calling function

function_name(parameters)

```
>> def sum(x,y):
    print("The summation is:",x+y)
>> sum(3,6)
```

The summation is: 9

Functions

Default Arguments

- Default values are assumed to be present if not explicitly provided

function_name(parameters)

```
>> def sum(a,b,x=3,y=9):
   print("The summation is:",x+y+a+b)
>> sum(1,2,7,8)
The summation is: 18
>> sum(1,2,7)
The summation is: 19
>> sum(1,2)
 The summation is: 15
```

Functions

Keyword Arguments

- These are special arguments(parameters) where parameter name are identified at the place of calling function.

function_name(parameters)

```
>> def sum(a,b,x,y):
    print("The summation is:",x+y+a+b)
>> sum(a=10,b=3,x=6,y=7)
The summation is: 26
>> sum(10,3,x=20,y=10)
The summation is: 43
```

Functions

Variable Arguments

- A Function designed to receive any number of arguments

function_name(parameters)

```
>> def sum(*x):
   s=0
   for i in x: s = s+i
   print(s)
>> sum(2,3,4)
9
>> sum(3,5)
8
```

Functions

Returning from function

- When function gets called, it returns automatically when execution is done by default. We can change by returning with some values if needed.

```
>>def sum(*x):
    s=0
    for i in x: s = s+i
    return s
>> s = sum(2,3,4)
     s
9
```

Functions

Returning Collection from function

- Returning List
- When function gets called, it returns automatically when execution is done by default. We can change by returning collection of values if needed, can be done with either list or dictionary.

```
>> def calc(x,y):
    sum = [x+y,x-y,x*y,x/y]
    return sum
>> res = calc(5,2)
print(res)
[7, 3, 10, 2.5]
```

Functions

Returning Dictionary

```
>> def calc(x,y):
    sum = {'Sum':x+y, 'Diff':x-y, 'Multiply':x*y, 'Division':x/y}
    return sum
>> result = calc(6,3)
     print(result)
{'Sum': 9, 'Diff': 3, 'Multiply': 18, 'Division': 2.0}
```

Functions

Global Variables

```
>> x=10
   def calc(x,y):
            sum = {'Sum':x+y, 'Diff':x-y, 'Multiply':x*y, 'Division':x/y}
            return sum
   result = calc(6,3)
   print(result)
   print(x)
```

```
{'Sum': 9, 'Diff': 3, 'Multiply': 18, 'Division': 2.0}
10
```

Functions

Nested Functions

```
>> def f():
    print("Inside f function")
    def g():
        print("Inside g function")
    g()

>> f()
  Inside f function
  Inside g function
>> g()
            NameError
```

Functions

Lambda Expressions

- To create anonymous function lambda expression is used
- Syntax

lambda parameters : expression

```
>> (lambda x: x**2)(4)
16
```

File Handling

- Introduction to File handling
- Opening and Closing Files
- Reading from Text Files
- Writing to Text Files
- Seeking Within Files
- Reading from CSV/XLS Files
- Writing to CSV/XLS Files

File Handling

- Introduction to File handling
- **Opening and Closing Files**
- Reading from Text Files
- Writing to Text Files
- Seeking Within Files
- Reading from CSV/XLS Files
- Writing to CSV/XLS Files

File Handling

- Logical Steps in File Handling

1. Open the file
2. Access the file
3. Close the file

File Handling

- Opening and Closing Files
- Opening the file : It requires 2 piece of information

1. Pathname of file
2. Access mode

r	Open the file for reading
w	Open the file for writing
a	Open the file for appending

File Handling

- Syntax

  Var = open(“filename.extn”, ”mode”)

- Example

```
>> f = open(“File.txt”, “r”)
>> f
<_io.TextIOWrapper name='File.txt' mode='r' encoding='cp1252'>
>> f = open("ile.txt",'r')
>> f
FileNotFoundError
```

File Handling

- Syntax

  f.close()

- Example

```
>> f = open(“File.txt”, “r”)
>> f
<_io.TextIOWrapper name='File.txt' mode='r' encoding='cp1252'>
>> f.close()
>> f
<_io.TextIOWrapper name='File.txt' mode='r' encoding='cp1252'>
>> f.closed
True
```

References

1. https://www.python.org/
2. Book: Learning Python Author: B Nagesh Rao ISBN-10:8193392329


thank you
danke
рахмат
謝謝
ngiyabonga
teşekkür ederim
Баярлалаа
спасибо
faafetai lava
mersi
barka
welalin
tack
dank je
misaotra
matondo
paldies
grazzi
gracias
vinaka
спасибі
blagodaram
kiitos
dankie
dhanyavad
enkosi
bedankt
dziękuje
hvala
maururu
köszönöm
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
хвала
asante
manana
obrigada
djiere dieuf
mochchakkeram
дякую
mamnun
sobodi
dėkuji
obrigado
mèsi
sukriya
kop khun krap
arigatô
dakujem
trugarez
ありがとう
tanemirt
rahmet
grazie
najis tuke
terima kasih
kam sah hamnida
diolch
dhanyavadagalu
shukriya
мерси
감사합니다
xièxie
ευχαριστώ
merci