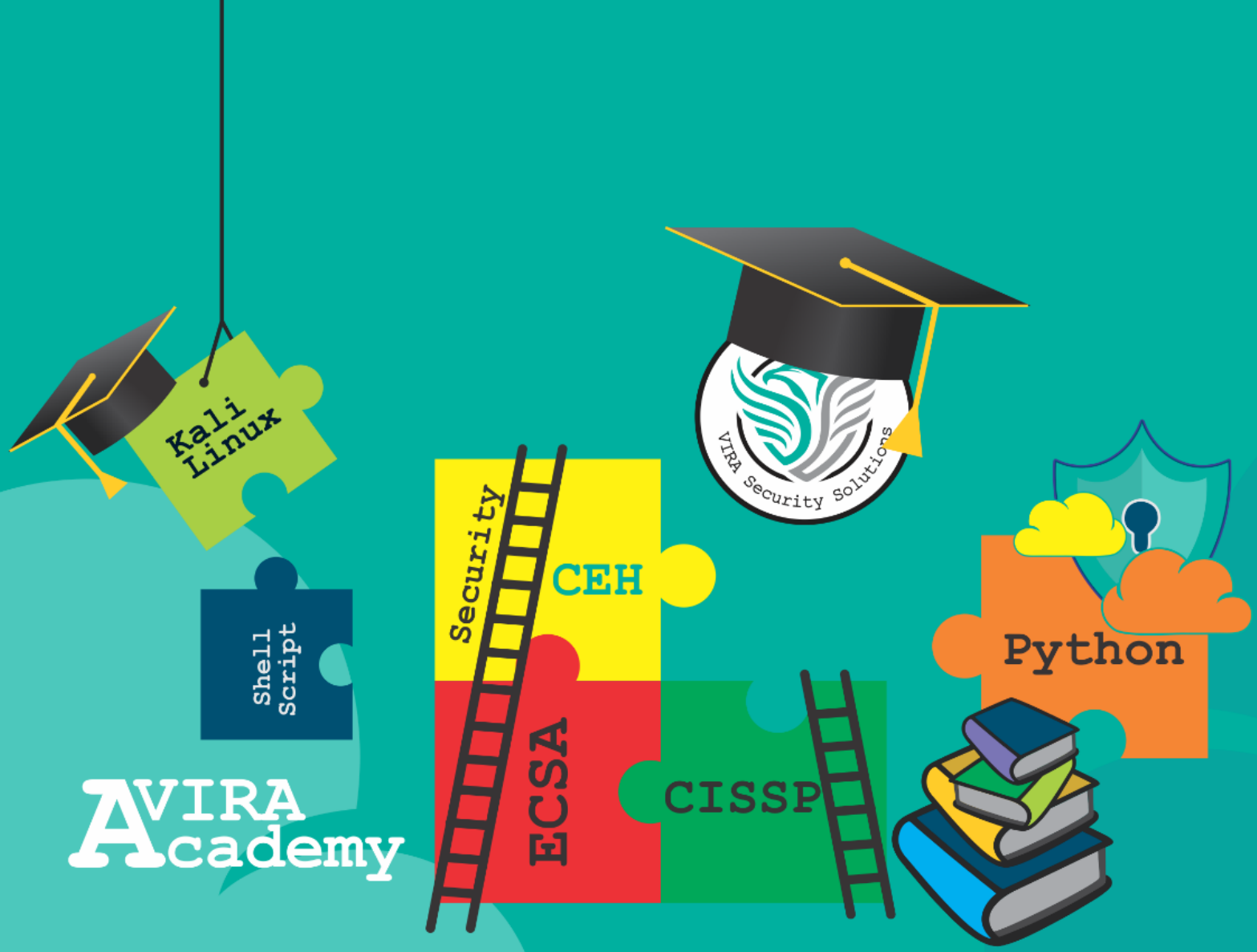

VIRA
Security
Solutions


Kali Linux
Shell Script
VIRA Academy
Security
CEH
ECSA
CISSP
VIRA Security Solutions
Python

Python

Part2

Mohammad Reza Gerami
Mrgerami@aut.ac.ir
gerami@virasec.ir

Augmented Assignment statements

Augmented assignment is the combination, in a single statement, of a binary operation and an assignment statement:

```
augmented_assignment_stmt: target augop expression_list
augop:          "+=" | "-=" | "*=" | "/=" | "%=" | "**="
              | ">>=" | "<<=" | "&=" | "^=" | "|="
target:         identifier | "(" target_list ")" | "[" target_list "]"
              | attributeref | subscription | slicing
```

Augmented Assignment statements

The basic syntax of the augmented assignments is an expression in which the same variable name appears on the left and the right of the assignment's. Now we have seen the example with the use of addition augmented assignment (+=) statement as:

```
total = 0
for number in [1, 2, 3, 4, 5]:
        total += number      # add number to total
print(total)
```

Output: 15

Addition Augmented Assignment

(+=)

This function (x += y) mathematically written as x = x+y.

Example :

```
x=0
x+=2
print(x)
```

Output: 2

Subtraction Augmented assignment

(-=)

This function (x -= y) mathematically written as x = x-y.

Example :

```
x=5
x-=2
print(x)
```

Output: 3

Multiplication Augmented assignment

(*=)

This function (x *= y) mathematically written as x = x*y.

Example :

```
x=2
x*=2
print(x)
```

Output: 4

Division Augmented assignment

(/=)

This function (x /= y) mathematically written as x = x/y.

Example :

```
x=10
x/=2
print(x)
```

Output: 5

Exponential Augmented assignment

(**=)

This function (x **= y) mathematically written as x = xy**

Example :

```
x=5
y=2
x**=y
print(x)
```

Output: 25

Python String

In Python, **Strings** are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

Creating a String

Strings in Python can be created using single quotes or double quotes or even triple quotes.

Python String


```
String1 = 'Welcome to the new World'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String
# with double Quotes
String1 = "I'm a pro"
print("\nString with the use of Double Quotes: ")
print(String1)
```

```
# Creating a String
# with triple Quotes
String1 = '''I'm a pro and I live in a world of
“professionals"""
print("\nString with the use of Triple Quotes: ")
print(String1)

# Creating String with triple
# Quotes allows multiple lines
String1 = ‘‘Python
            For
          better Life'''
print("\nCreating a multiline String: ")
print(String1)
```

Accessing characters in Python

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

While accessing an index out of the range will cause an IndexError. Only Integers are allowed to be passed as an index, float or other types will cause a TypeError.

Accessing characters in Python

P	y	t	h	o	n	f	o	r	P	r	o
0	1	2	3	4	5	6	7	8	9	10	11

```
String1 = "PythonforPro"
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])
```

String Slicing

To access a range of characters in the String, method of slicing is used. Slicing in a String is done by using a Slicing operator (colon).

```
# Creating a String
String1 = "PythonforPro"
print("Initial String: ")
print(String1)

# Printing 3rd to 12th character
print("\nSlicing characters from 3-12: ")
print(String1[3:12])

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

Deleting/Updating from a String

In Python, Updating or deletion of characters from a String is not allowed. This will cause an error because item assignment or item deletion from a String is not supported. Although deletion of entire String is possible with the use of a built-in del keyword. This is because Strings are immutable, hence elements of a String cannot be changed once it has been assigned. Only new strings can be reassigned to the same name.

```
String1 = "Hello, I'm a pro"
print("Initial String: ")
print(String1)

# Updating a character
# of the String
String1[2] = 'p'
print("\nUpdating character at 2nd Index: ")
print(String1)
```

Deleting/Updating from a String

Deletion of a character:

```
# Deleting a character
# of the String
del String1[2]
print("\nDeleting character at 2nd Index: ")
print(String1)
```

Deleting/Updating from a String

Deleting Entire String:

Deletion of entire string is possible with the use of del keyword. Further, if we try to print the string, this will produce an error because String is deleted and is unavailable to be printed.

```
# Deleting a String
# with the use of del
del String1
print("\nDeleting entire String: ")
print(String1)
```

Escape Sequencing in Python

While printing Strings with single and double quotes in it causes **SyntaxError** because String already contains Single and Double Quotes and hence cannot be printed with the use of either of these. Hence, to print such a String either Triple Quotes are used or Escape sequences are used to print such Strings.
Escape sequences start with a backslash and can be interpreted differently. If single quotes are used to represent a string, then all the single quotes present in the string must be escaped and same is done for Double Quotes.

Escape Sequencing in Python

```
String1 = '''I'm a "Pro"'''
print("Initial String with use of Triple Quotes: ")
print(String1)

# Escaping Single Quote
String1 = 'I\'m a "Pro"'
print("\nEscaping Single Quote: ")
print(String1)

# Escaping Doule Quotes
String1 = "I'm a \"Pro\""
print("\nEscaping Double Quotes: ")
print(String1)

# Printing Paths with the
# use of Escape Sequences
String1 = "C:\\Python\\Geeks\\"
print("\nEscaping Backslashes: ")
print(String1)
```

```
Initial String with use of Triple Quotes:
I'm a "Pro"

Escaping Single Quote:
I'm a "Pro"

Escaping Double Quotes:
I'm a "Pro"
```

Formatting of Strings

Strings in Python can be formatted with the use of format() method which is very versatile and powerful tool for formatting of Strings. Format method in String contains curly braces {} as placeholders which can hold arguments according to position or keyword to specify the order.

```
String1 = "{} {} {}".format('Pro', 'For', 'Life')
print("Print String in default order: ")
print(String1)

# Positional Formatting
String1 = "{1} {0} {2}".format('Pro', 'For', 'Life')
print("\nPrint String in Positional order: ")
print(String1)

# Keyword Formatting
String1 = "{l} {f} {g}".format(g = 'Pro', f = 'For', l = 'Life')
print("\nPrint String in order of Keywords: ")
print(String1)
```

Output:

```
Print String in default order:
Pro For Life

Print String in Positional order:
For Pro Life

Print String in order of Keywords:
Life For Pro
```

Formatting of Strings

Integers such as Binary, hexadecimal, etc. and floats can be rounded or displayed in the exponent form with the use of format specifiers.

```
# Formatting of Integers
String1 = "{0:b}".format(16)
print("\nBinary representation of 16 is ")
print(String1)

# Formatting of Floats
String1 = "{0:e}".format(165.6458)
print("\nExponent representation of 165.6458 is ")
print(String1)

# Rounding off Integers
String1 = "{0:.2f}".format(1/6)
print("\none-sixth is : ")
print(String1)
```

Output:

Binary representation of 16 is
10000

Exponent representation of 165.6458 is
1.656458e+02

one-sixth is :
0.17

Formatting of Strings

A string can be left() or center(^) justified with the use of format specifiers, separated by colon(:).

```
# String alignment
String1 = "|{:<10}|{:^10}|{:>10}|".format('Python','for','Everyone')
print("\nLeft, center and right alignment with Formatting: ")
print(String1)

# To demonstrate aligning of spaces
String1 = "\n{0:^16} was founded in {1:<4}!".format("PythonforEveryone", 2020)
print(String1)
```

```
Left, center and right alignment with Formatting:
|Python    |   for   |    Everyone|

PythonforEveryone   was founded in 2020 !
```

Formatting of Strings

Old style formatting was done without the use of format method by using **%** operator

```
 Integer1 = 12.3456789
print("Formatting in 3.2f format: ")
print('The value of Integer1 is %3.2f' %Integer1)
print("\nFormatting in 3.4f format: ")
```

```
Formatting in 3.2f format:
The value of Integer1 is 12.35

Formatting in 3.4f format:
The value of Integer1 is 12.3457
```

How Strings are Indexed

A string's characters correspond to an index number, starting with the index number 0

For the string "PythonforPro" the index breakdown looks like this:

P	y	t	h	o	n	f	o	r	P	r	o
0	1	2	3	4	5	6	7	8	9	10	11

Strings Index

Old style formatting was done without the use of format method by using **%** operator

```
txt = "Hello, welcome to my world."

x = txt.index("welcome")

print(x)
```

Output:
7

Strings Index

Syntax:

string.index(*value, start, end*)

```
txt = "Hello, welcome to my world."

x = txt.index("e", 5, 10)

print(x)
```

Mutable vs Immutable Objects

All the data in a Python code is represented by objects or by relations between objects. Every object has an identity, a type, and a value

Identity — An object's identity never changes once it has been created; you may think of it as the object's address in memory. The is operator compares the identity of two objects; the id() function returns an integer representing its identity.

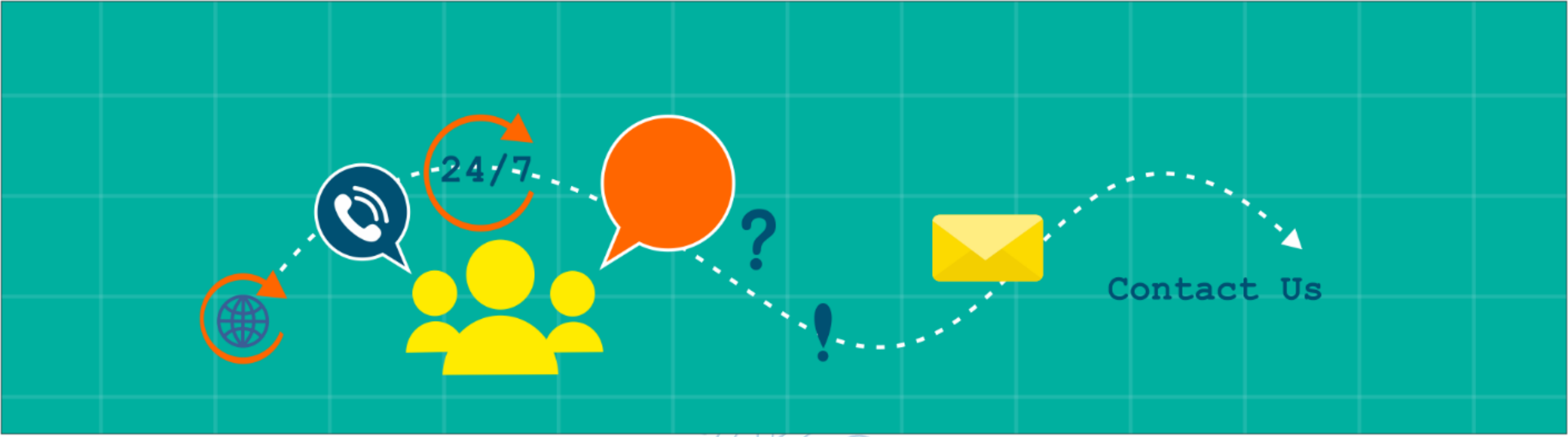
Type — An object's type defines the possible values and operations (e.g. "does it have a length?") that type supports. The type() function returns the type of an object. An object type is unchangeable like the identity.

Value — The value of some objects can change. Objects whose value can change are said to be mutable; objects whose value is unchangeable once they are created are called immutable

Mutable vs Immutable Objects

```
list_values = [1, 2, 3]
set_values = (10, 20, 30)
print(list_values[0])
print(set_values[0])
```

```
list_values = [1, 2, 3]
set_values = (10, 20, 30)
list_values[0] = 100
print(list_values)
set_values[0] = 100
```



Visiting Address: No 53 Vafa Manesh Ave

Heravi, Pasdaran Ave, TEHRAN-IRAN

Tel No: 0098 21 22196115-09125792641

Email: info@ virasec.ir

Website: www.virasec.ir

آدرس: تهران، پاسداران، هروی، خیابان وفامنش، پلاک ۵۳

شماره تماس: ۰۹۱۲۵۷۹۲۶۴۱-۰۲۱۲۲۱۹۶۱۱۵