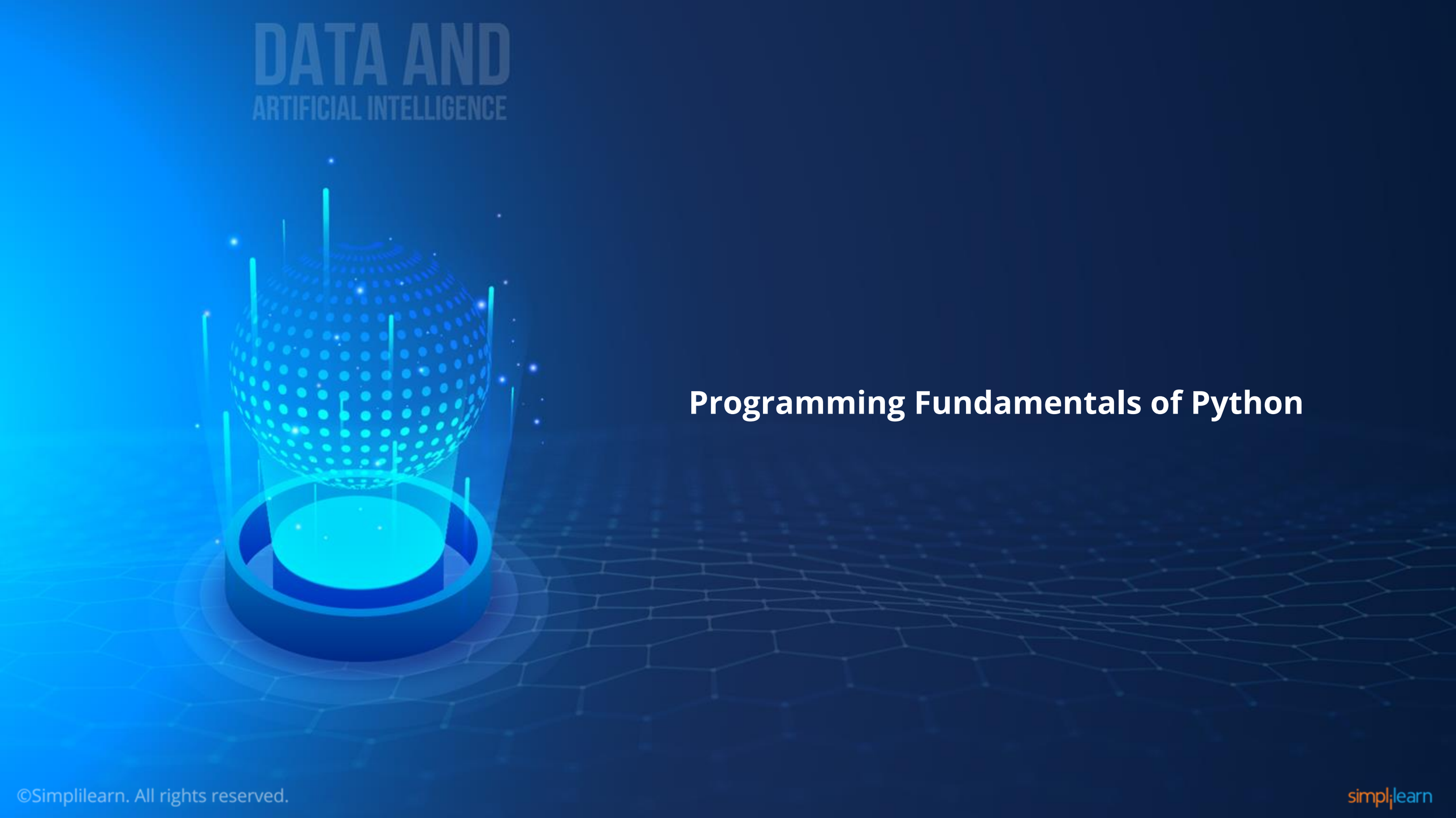
Programming Basics and Data Analytics with Python

simplilearn



Programming Fundamentals of Python

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- Implement variables, data types, keywords, and expressions
- Use operators, functions, conditions, and branching in Python programs
- Create Python programs using string operations, tuples, lists, sets, and dictionaries
- Construct loops in Python programs

Variables

Variables

Variables are used to store data in a computer's memory.

Example

```
price= 30
print(price)
```

Output: 30

A variable type is assigned with a data type.

Example

```
type ('message')
```

Output: str

```
type(10)
```

Output: int

Variable Names

Conditions for a variable name:

- Can be long and meaningful
- Can contain letters and numbers but should not begin with a number
- Can have an underscore character

An illegal name to a variable will result in a syntax error.

Examples

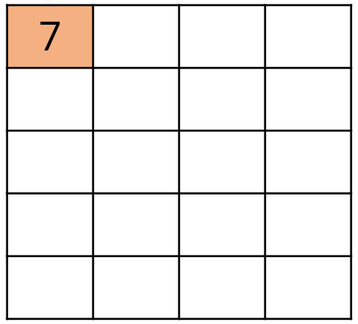
```
76trombones = 'big parade'
SyntaxError: invalid syntax
more@ = 1000000
SyntaxError: invalid syntax
class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

Assignment and Reference

When a variable is assigned a value, it refers to the value's memory location or address. It is not equal to the value.

```
x = 7
```

Ref: <address 1>



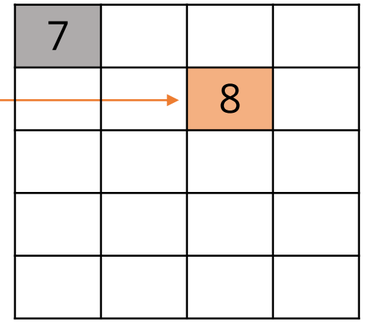
Memory location

```
x = 7
x = x + 1

print(x)
```

```
8
```

Ref: <address 12>

Garbage collected



Memory location

Variable Assignment

A variable can be assigned or bound to any value.

Some characteristics of binding a variable in Python are listed here.

```
In [1]: x = 3
        type(x)

Out[1]: int
```

The variable refers to the memory location of the assigned value.

```
In [2]: y = 2.1
        type(y)

Out[2]: float
```

The variable appears on the left, while the value appears on the right.

```
In [3]: z = 'test'
        type(z)

Out[3]: str
```

The data type of the assigned value and the variable are the same.

Example: Variable Assignment

Assigning a value to a variable and printing the variable and its data type

```
[1]: first_string_variable = 'test'
     first_integer_variable = 100
```

Assignment

```
[2]: print(first_string_variable)
     print(first_integer_variable)
```

```
test
100
```

Variable data value

```
[3]: print(type(first_string_variable))
     print(type(first_integer_variable))
```

```
<class 'str'>
<class 'int'>
```

Data type of the object

Multiple Assignments

You can access a variable only if it is assigned. Multiple variables can be assigned simultaneously.

```
In [48]: number_example
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-48-a856f233ae98> in <module>()
----> 1 number_example

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

Access variable without assignment

```
In [49]: number_example = 2
         number_example

Out[49]: 2
```

Access variable after assignment

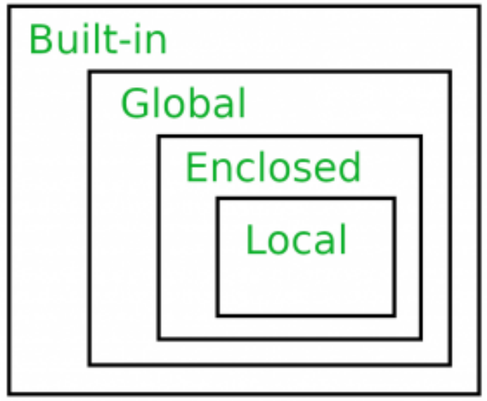
```
In [54]: integer_x, integer_y = 5,22

In [55]: integer_x

Out[55]: 5

In [56]: integer_y

Out[56]: 22
```

Multiple assignments

Scope of the Variable

Scope refers to the visibility of the variable. There are four types of variable scope:

- **Local**: Variables can only be accessed within its block.
- **Global:** Variables that are declared in the global scope can be accessed from anywhere in the program
- **Enclosed:** A scope that is not local or global comes under enclosing scope.
- **Built-in**: All reserved names in Python built-in modules have a built-in scope.



Scope of the Variable

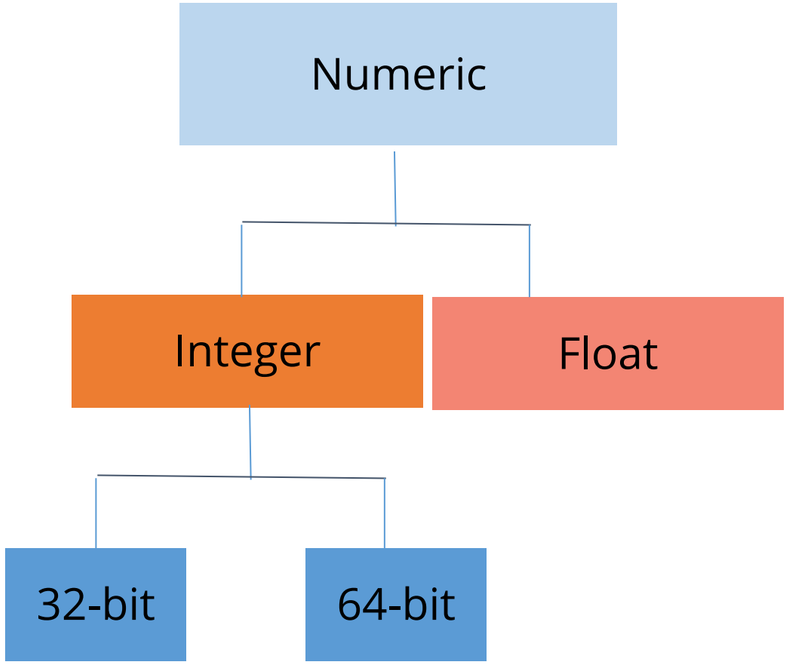
Local Scope	Global Scope	Enclosing Scope	Built-in Scope

`# Local Scope`
 	
`i = 'global variable'`
`def inner():`
`  i = 'inner variable'`
`  print(i)`
 	
`inner()`
 	
`inner variable`
	`# Global Scope`
	
	`i = 'global variable'`
	`def inner():`
	`  pi = 'inner variable'`
	`  print(i)`
	
	`inner()`
	`print(i)`
	
	`global variable`
	`global variable`
		`# Enclosed Scope`
		`i = 'global variable'`
		`def outer():`
		`  i = 'outer variable'`
		`  def inner():`
		`  # i = 'inner variable'`
		`  nonlocal i`
		`  print(i)`
		`  inner()`
		
		`outer()`
		`print(i)`
		
		`outer variable`
		`global variable`
			`a = 5.5`
			`int(a)`
			`print(a)`
			`print(type(a))`
			
			`5.5`
			`<class 'float'>`

Data Types in Python

Basic Data Types: Integer and Float

Python supports various data types. There are two main numeric data types:




```
In [9]:  int_number = 7/2
         int_number

Out[9]:  3
```

Integer value

```
In [10]: float_number = 5.2/2
         float_number

Out[10]: 2.6
```

Float value

Basic Data Types: String

Python has extremely powerful and flexible built-in string processing capabilities.

```
In [14]: string_one = 'first string'
         string_two = "second string"
         string_three = """third string"""
```

With single quotes

With double quotes

With three double quotes

```
In [15]: print(string_one)
         print(string_two)
         print(string_three)

         first string
         second string
         third string
```

Print string values

Basic Data Types: Null and Boolean

Python also supports Null and Boolean data types.

```
In [102]: num_x = None
          num_x is None
Out[102]: True
```

Null value type

Boolean type

```
In [103]: num_x = 10
          num_x is None
Out[103]: False
```

Boolean type

Type Casting

You can change the data type of any variable using type casting.

```
In [58]: float_number = 3.6467
```

Float number

```
In [59]: float_number
Out[59]: 3.6467
```

```
In [60]: int(float_number)
Out[60]: 3
```

Type cast to integer

```
In [61]: str(float_number)
Out[61]: '3.6467'
```

Type cast to string value

Data Types in Python

Problem Statement: Write a program to print and identify the an integer data type.

Steps to Perform:

1. Enter the data
2. Print the data type

ASSISTED PRACTICE

Keywords and Identifiers

Keywords

Keywords are reserved words in Python.

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	-
class	exec	in	raise	-
continue	finally	is	return	-
def	for	lambda	try	-

Identifiers

An identifier is a name given to entities such as classes, functions, and variables that differentiate one entity from another.

Rules for writing identifiers:

- Can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _

  For example: myClass, Simpli_123
- Cannot start with a digit

  For example: var123
- Keywords cannot be used as identifiers
- Cannot use special symbols such as !, @, #, $, and %
- Can be of any length

Points to Remember

- Python is a case-sensitive language.

  Example: Variable or variable

- Variable declaration should make sense.

  Example: c = 10 or count = 10

- Multiple words can be separated using an underscore.

  Example: this_is_a_long_variable

Expressions

Expressions

An expression is a combination of values, variables, and operators. Individual values and variables are also considered expressions.

Examples of legal expressions:

17

x

x + 17

Conditional Expressions

They are used for comparison.

Conditional statements supported by Python:

Equal to a==b	Not equal to a!=b	Less than a<b
Less than or equal to a<=b	Greater than a>b	Greater than or equal to a>=b

Membership Expressions

Membership expressions validate the membership in a sequence such as strings, lists, or tuples.

The different membership expressions in Python:

```
In [14]: a="learning is an art"
         b="art"
         b in a

Out[14]: True

In [15]: b is a

Out[15]: False

In [16]: b is "art"

Out[16]: True
```

Basic Operators

Arithmetic Operators

These operations (operators) can be applied to all numeric types.

Operator	Description	Example
+, -	Addition and subtraction	10 + 3 = 13 40 - 14 = 26
*,%	Multiplication and modulo	2 * 3 = 6 27 % 5 = 2
/	Division	10 / 3 = 3.3333333 (Python 3) 10 / 3 = 3 (Python 2)
//	Truncation division	10 // 3 = 3 10.0 // 3 = 3.0
**	Power of number	2 ** 3 = 8(2 to power 3)

Assignment Operator

- ”=“ is used to assign a value to a variable.

Example

```
x = 20
```

- Python allows the assignment of multiple variables in a single line.

Example

```
a, b = 10, 20
```

The expressions on the right-hand side are evaluated before any assignment occurs.

- The evaluation order of an expression is from left to right.

Example

```
a,b = 1,2
a,b = b, a+b
a
2
b
3
```

Comparison Operator

Comparison operators include <, <=, >, >=, !=, and ==.

Example

```
a = 20
b = 30
print(a>b)
 False
```

Logical Operator

- And, or, and not (&&, ||, and !) are the logical operators.

A	B	A and B	A or B	!A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

- Consider A = a%3==0 and B = a%5==0. When logical operators are applied, they are evaluated based on the evaluation of expressions.

Example

```
a = 25
print(a%3==0 and a%5==0)
False
```

Bitwise Operator

|, &, ^, and ~ (Bitwise Or, Bitwise And, Bitwise XOR, and Bitwise Negation) are the bitwise operators.

Example

a = 2 (010)

b = 3 (011)

a & b = 2

a | b = 3

a ^ b = 1

A	B	A & B	A | B	A ^ B
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Operators in Python

Problem Statement: Write a program to insert three sides of a triangle and check whether it is an isosceles triangle or not.

Steps to Perform:

1. Enter the variables
2. Define conditions to identify whether the triangle is isosceles or not

ASSISTED PRACTICE

Functions

Functions

Functions are the most important aspects of an application.
It is defined as the organized block of reusable code.

Syntax
def <name>(arg1, arg2, ..., argN):   <statements>   return <value>

Properties
• Outcome of the function is communicated by a return statement. • Arguments in parenthesis are basically assignments.

Use **def** to create a function and assign a name to it.

Functions: Considerations

Some points to consider when defining functions:

- A function should always have a return value
- If **return** is not defined, then it returns **None**
- Function overloading is not permitted

Function name

Create function

Argument

```
def my_first_function(name):
 return name
print (my_first_function('Jack'))
```

Return type

```
Jack
```

Call function

Functions: Returning Values

You can use a function to return a single value or multiple values.

In [256]:
```
def add_two_numbers(num1, num2):
    return num1+num2

number1 = 23
number2  = 47.5
result = add_two_numbers(number1,number2)
result
```

Create function

Call function

Out[256]:
```
70.5
```

In [257]:
```
def profile():
    age = 21
    height = 5.5
    weight = 130
    return age, height, weight

age, height, weight = profile()
```

Create function

Multiple return

Call function

In [258]:
```
print (age, height, weight)
```

```
21 5.5 130
```

Built-in Sequence Functions

Enumerate
Indexes data to keep track of indices and corresponding data mapping

Sorted
Returns the new sorted list for the given sequence

Reversed
Iterates the data in reverse order

Zip
Creates lists of tuples by pairing up elements of lists, tuples, or other sequence

Built-in Sequence Functions: Enumerate

```
short_list =  ['McDonald', 'Taco Bell', 'Dunkin', 'Wendys', 'Chiptole']
for position,name in enumerate(short_list):
    print (position, name)
```

List of food stores

```
0 McDonald
1 Taco Bell
2 Dunkin
3 Wendys
4 Chiptole
```

Print data element and index using enumerate method

```
store_map=dict((name,position) for position,name in enumerate(short_list))
```

Create a data element and index map using dict

```
store_map
```

```
{'McDonald': 0, 'Taco Bell': 1, 'Dunkin': 2, 'Wendys': 3, 'Chiptole': 4}
```

View the store map in the form of key-value pair

Built-in Sequence Functions: Sorted

```
In [27]: sorted([91,43,65,56,7,33,21])
Out[27]: [7, 21, 33, 43, 56, 65, 91]
```

Sort numbers

```
In [28]: sorted('the data science')
Out[28]: [' ',
          ' ',
          ' ',
          'a',
          'a',
          'c',
          'c',
          'd',
          'e',
          'e',
          'e',
          'h',
          'i',
          'n',
          's',
          't',
          't']
```

Sort a string value

Built-in Sequence Functions: Reversed and Zip

```
[1]: num_list = range(15)
     list(reversed(num_list))
```

Create a list of numbers for range 15

```
[1]: [14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Use reversed function to reverse the order

```
[5]: subjects= ['math','staticstics','algebra']
     subject_count=['one','two','three']
```

Define list of subjects and count

```
[12]: total_subject= zip(subjects,subject_count)
      total_subject= list(total_subject)
      print(total_subject)

      [('math', 'one'), ('staticstics', 'two'), ('algebra', 'three')]
```

Zip function to pair the data elements of lists

```
[13]: type(total_subject)
```

Return list of tuples

```
[13]: list
```

View type

Search for a Specific Element from a Sorted List

Problem Statement: Data of a company is stored in a sorted list. Write a program to search for a specific element from the list.

Steps to Perform:

1. Enter the sorted list
2. Enter the element to be searched
3. Use an if...else statement to search for the element

ASSISTED PRACTICE

Create a Banking System Using Functions

Problem Statement: Design a software for a bank. The software should have options such as cash withdrawal, cash credit, and change password. The software should provide the required output according to the user input.

Hint: Use if...else statements and functions

Steps to Perform:

1. Declare the variables
2. Create a function to perform the functions (withdrawal, cash credit, and change password)
3. Use if...else statement to provide the required output

UNASSISTED PRACTICE

Unassisted Practice: Create a Bank System Using Functions

```
[1]: Total=0
current=50000  # current balance

def withdraw():
    with_amount=input("Enter withdraw amount")
    Total= current-int(with_amount)
    print("Your account balance is: ",Total)
    return()

def credit():
    credit_amount=input("Enter amount to be credited")
    Total=current+int(credit_amount)
    print("Your account balance is: ", Total)
    return()


def change_pass():
    old=input("Enter old password")
    new=input("Enter new password")
    print("You password is changed: ",new)
```

Initial balance in the bank account

Function defined to withdraw an amount

Function defined to credit an amount

Function defined to change the password of an account

Unassisted Practice: Create a Bank System Using Functions

```
acco_no=input("Enter your account number")
choice=input("Enter your choice:\n 1: Cash withdraw \n 2: Cash Credit \n 3: Change password")

if(choice=='1'):
    withdraw()

elif(choice=='2'):
    credit()

else:
    change_pass()
```

```
Enter your account number 1234567
Enter your choice:
 1: Cash withdraw
 2: Cash Credit
 3: Change password 2
Enter amount to be credited 2000
Your account balance is:  52000
```

Output

String Operations

String in Python

A string is a:

- Sequence of characters
- Sequence of "pure" unicode characters (there is no specific encoding like UTF-8)

There are different ways to define strings in Python.

Examples

astring = "Hello world!“ *#double quotes*

astring2 = 'Hello world!’ *#single quotes*

astring3 = ‘’’ Hello world! ’’’*#three single quotes*

astring4 = “”” Hello world! “”””*#three double quotes*

String Functions: Concatenation, Repetition, and Indexing

Concatenation

Strings can be glued together (concatenated) with the + operator.

Example

```
Print(“Hello” + “World”)
```

Output: HelloWorld

Repetition

Strings can be repeated or repeatedly concatenated with the asterisk operator "*".

Example

```
Print("*-*" * 3)
```

Output: *-**-**-*

Indexing

A string can be indexed using index() method.

Example

```
astring = "Hello world!“
print(astring.index("o"))
```

Output: 4

Access Characters in Strings

A string in Python:

- Consists of a series of characters such as letters, numbers, and special characters
- Can be subscripted or indexed (Similar to C, the first character of a string in Python has the index 0.)

Example

```
astring = "Hello world“
print(astring[0])
#The last character of a string can be accessed like this:
print(astring[len(astring)-1])
print(astring[-2])
```

Output:

H
d
l

-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
H	e	l	l	o	w	o	r	l	d	
0	1	2	3	4	5	6	7	8	9	10

String Functions: Slicing

- Substrings can be created with the slice or slicing notation, that is, two indices in square brackets separated by a colon.

Example

```
Print("Python"[2:4])
```

Output: th

- Size of a string can be calculated using len().

Example

```
len("Python") will result in 6
```

- Extended slice syntax can be used to create a substring with skipping indices in between the string.

Example

```
astring = "Hello world!"
print(astring[3:7:2])
```

Output: l

String Functions: Uppercase and Lowercase

Strings can be converted to uppercase and lowercase, respectively.

Example

```
astring = "Hello world!“
print(astring.upper())
print(astring.lower())
```

Output: HELLO WORLD!

hello world!

String Functions: Startswith and Split

- The startswith() method returns *True* if a string starts with the specified prefix(string). Else, it returns *False*.

Example

```
astring = "Hello world!“
print(astring.startswith("Hello"))
print(astring.endswith("asdfasdfasdf"))
```

Output: True
False

- The split() method breaks up a string at the specified separator and returns a list of strings.

Example

```
astring = "Hello world!“
afewwords = astring.split(" ")
print(afewwords)
```

Output: [‘Hello’, ‘world!’]

Immutable Strings

Python strings cannot be changed as change in indexed position will raise an error.

Example

```
astring = "Some things are immutable!”
astring[-1] = “.”
```

Output:

```
Traceback (most recent call last):
        File "<stdin>", line 1, in <module>
      TypeError: 'str' object does not support item assignment
```

Escape Sequences

The backslash (\) character is used to escape characters, that is, to "escape" the special meaning which the character would otherwise have.

Escape Sequence	Meaning Notes
\newline	Ignored
\\	Backslash (\)
\’	Single quote (')
\"	Double quote (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\N{name}	Character named name in the Unicode database (Unicode only)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\uxxxx	Character with 16-bit hex value xxxx (Unicode only)
\Uxxxxxxxx	Character with 32-bit hex value xxxxxxxx (Unicode only)

String Operations in Python

Problem Statement: An employee at ABC Inc. is tasked with writing a program that capitalizes the first and last letter of a each word in a string.

Steps to Perform:

1. Enter the string
2. Use a loop to perform the operation
3. Print the string

ASSISTED PRACTICE

Tuples

Tuple

A tuple is a one-dimensional, immutably ordered sequence of items that can be of mixed data types.

```
In [145]: first_tuple = (12,'Jack',45.6,'new',(3,2),'test')
```

Create a tuple

```
In [146]: first_tuple
Out[146]: (12, 'Jack', 45.6, 'new', (3, 2), 'test')
```

View tuple

```
In [147]: first_tuple[1]
Out[147]: 'Jack'
```

Access the data at index value 1

```
In [148]: first_tuple[1] = 'Mark'
```

Try to modify the tuple

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-148-38afcbb40e37> in <module>()
----> 1 first_tuple[1] = 'Mark'

TypeError: 'tuple' object does not support item assignment
```

Error: A tuple is immutable and cannot be modified.

Accessing Tuples

You can access a tuple using indices.

```
In [1]: first_tuple = (12, 'Jack', 45.6, 'new', (3,2), 'test')
```

Tuple

```
In [2]: #Accessing elements using a positive index
        #The index count starts from the left, with the first index being 0
        first_tuple[2]

Out[2]: 45.6
```

Access with positive index

```
In [3]: #Accessing elements using a negative index
        #The index count starts from the right, with the first index being -1
        first_tuple[-3]

Out[3]: 'new'
```

Access with negative index

Slicing Tuples

You can also slice a range of elements by specifying the start and end indices of the desired range.

```
In [1]: first_tuple = (12, 'Jack', 45.6, 'new', (3,2), 'test')
```

Tuple

```
In [4]: #Creating a subset/slice of the tuple
        #Specify the indices of the elements, separated by a colon
        #The first index is inclusive; the second index is exclusive
        first_tuple[1:4]

Out[4]: ('Jack', 45.6, 'new')
```

The count starts with the first index but stops before the second index.

```
In [5]: #You can use negative indices as well to slice a tuple
        #Count from the right, starting from -1, to specify the correct index
        first_tuple[1: -1]

Out[5]: ('Jack', 45.6, 'new', (3, 2))
```

The count stops before the second index for negative indices too.

Tuples in Python

Problem Statement: A trainer has requested his trainees to create a tuple with a repetition and test slicing from the end of the tuple.

Steps to Perform:

1. Perform a tuple with repetition
2. Print the repetition tuple
3. Perform slicing

ASSISTED PRACTICE

Lists

List

A list is a one-dimensional, mutably ordered sequence of items that can be of mixed data types.

```
In [161]: first_list = ['Mark',101,23.6,'test',None,11]
```

Creates a list

```
In [162]: first_list
Out[162]: ['Mark', 101, 23.6, 'test', None, 11]
```

Views a list

```
In [163]: first_list.append('Jack')
          first_list
Out[163]: ['Mark', 101, 23.6, 'test', None, 11, 'Jack']
```

Modifies a list: Add new items

```
In [164]: first_list.remove('Mark')
          first_list
Out[164]: [101, 23.6, 'test', None, 11, 'Jack']
```

Modifies a list: Remove items

```
In [165]: first_list.pop(2)
Out[165]: 'test'
```

Accesses and removes list data using element indices

```
In [166]: first_list.insert(1,'Smith')
          first_list
Out[166]: [101, 'Smith', 23.6, None, 11, 'Jack']
```

Modifies a list: Insert a new item at a certain index

Accessing Lists

```
In [5]: first_list

Out[5]: [101, 'Smith', 'Smith', 23.6, None, 11, 'Jack']
```

New modified list

```
In [6]: #Accessing elements using a positive index
        #The index count starts from the left, with the first index being 0
        first_list[2]

Out[6]: 'Smith'
```

Access with positive index

```
In [7]: #Accessing elements using a negative index
        #The index count starts from the right, with the first index being -1
        first_list[-2]

Out[7]: 11
```

Access with negative index

Slicing Lists

Slicing works with the elements in a list using indices.

```
In [5]: first_list

Out[5]: [101, 'Smith', 'Smith', 23.6, None, 11, 'Jack']
```

New modified list

```
In [8]: #Creating a subset/slice of the tuple
        #Specify the indices of the elements, separated by a colon
        #The first index is inclusive; the second index is exclusive
        first_list[1:4]

Out[8]: ['Smith', 'Smith', 23.6]
```

The count starts with the first index but stops before the second index.

```
In [9]: #You can use negative indices as well to slice a tuple
        #Count from the right, starting from -1, to specify the correct index
        first_list[1:-1]

Out[9]: ['Smith', 'Smith', 23.6, None, 11]
```

The count stops before the second index for negative indices too.

Lists in Python

Problem Statement: Write a program to take a string from the middle with some positional gap between characters.

Steps to Perform:

1. Enter the list
2. Print the list
3. Perform slicing

ASSISTED PRACTICE

Sets

Sets

Sets can be created by using the built-in **set()** function with an iterable object or a sequence. The sequence should be placed inside curly braces and must be separated by commas.

Examples

```
a = set([1, 2, 3, 4])
b = set([3, 4, 5, 6])
a | b # Union
{1, 2, 3, 4, 5, 6}
a & b # Intersection
{3, 4}
a < b # Subset
False
a - b # Difference
{1, 2}
a ^ b # Symmetric Difference
{1, 2, 5, 6}
```

Adding Elements in Sets

Using add() method

- Elements can be added to the set by using built-in **add()** function.

Using update() method

- For addition of two or more elements, **update()** method is used.

Examples

Using add():

```
set() #Initial blank Set
{8, 9, (6, 7)} #Set after addition of three elements
{1, 2, 3, (6, 7), 4, 5, 8, 9} #Set after addition of elements from 1-5
```

Using update():

```
set1 = set([ 4, 5, (6, 7)])
set1.update([10, 11])
{10, 11, 4, 5, (6, 7)} #Set after addition of elements using update
```

Accessing a Set

Sets are unordered items that have no index. Set items can be looped using a *for loop*.
If a specified value is present in a set, it can be looped by using the *in* keyword.

```
[2]: set1 = set(["abc", "c", "abc"])
     print("\nInitial set")
     print(set1)

     # Accessing element using
     # for loop
     print("\nElements of set: ")
     for i in set1:
         print(i, end=" ")

     # Checking the element
     # using in keyword
     print("abc" in set1)
```

Removing Elements from the Set

Using the remove() method or discard() method

- Elements can be removed from the set by using the *remove()* function. However, a KeyError arises if the element doesn't exist in the set.
- To remove elements from a set without KeyError, use the *discard()* function. If the element doesn't exist in the set, it remains unchanged.

Example

Initial Set: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Set after Removal of two elements: {1, 2, 3, 4, 7, 8, 9, 10, 11, 12}

Set after Discarding two elements: {1, 2, 3, 4, 7, 10, 11, 12}

Set after Removing a range of elements: {7, 10, 11, 12}

Frozen Sets

The frozenset() method returns an immutable frozenset object initialized with elements from the given iterable.

Example

```
String = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h') #Creating a set
frozenset(String)
frozenset() #To print empty frozen set
```

Set Methods

FUNCTION	DESCRIPTION
add()	Adds an element to a set
remove()	Removes an element from a set. If the element is not present in the set, raise a KeyError
clear()	Removes all elements form a set
copy()	Returns a shallow copy of a set
pop()	Removes and returns an arbitrary set element. Raise KeyError if the set is empty
update()	Updates a set with the union of itself and others
union()	Returns the union of sets in a new set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from set if it is a member. (Do not use the discard function if element is not in set)
intersection()	Returns the intersection of two sets as a new set

Sets in Python

Problem Statement: You are given an assignment to write a program to perform different set operations. You should also provide comments explaining each step.

Steps to Perform:

1. Create two sets
2. Display two sets
3. Find union and intersection
4. Check the relation between union and intersection sets
5. Display the relation between union and intersection sets
6. Difference between union and intersection sets

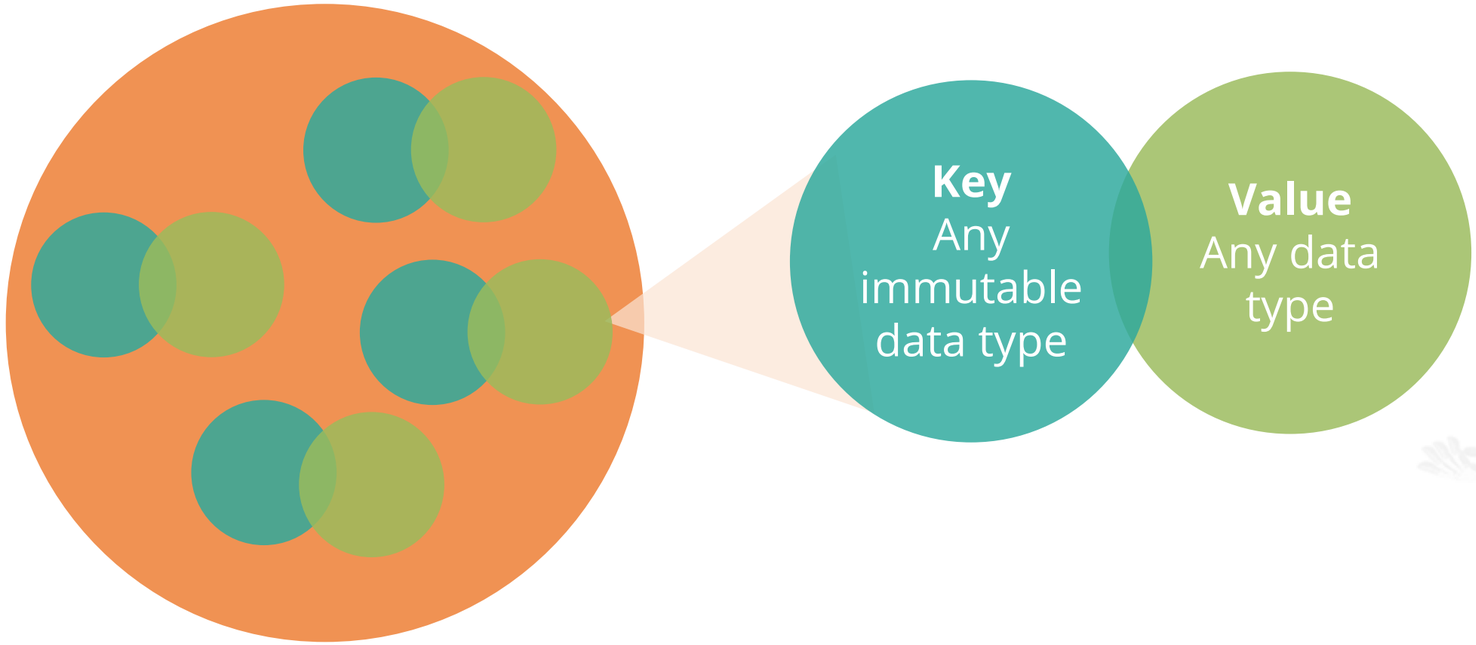
ASSISTED PRACTICE

Dictionaries

Dictionary

Dictionaries store a mapping between a set of keys and a set of values.

Dictionary



Define | Modify | View | Lookup | Delete

View Dictionaries

You can view the keys and values in a dictionary, either separately or together, using the syntax shown below:

```
In [215]: first_dict = {'John':'john@abc.com','Kelly':'kelly@xyz.org','id':[23,81]}
```

Creates a dictionary

```
In [216]: first_dict
Out[216]: {'John': 'john@abc.com', 'Kelly': 'kelly@xyz.org', 'id': [23, 81]}
```

Views entire dictionary

```
In [217]: first_dict.keys()
Out[217]: ['Kelly', 'John', 'id']
```

Views only keys

```
In [218]: first_dict.values()
Out[218]: ['kelly@xyz.org', 'john@abc.com', [23, 81]]
```

Views only values

Access and Modify Dictionary Elements

You can also access and modify individual elements in a dictionary.

```
In [219]: first_dict['Kelly']
Out[219]: 'kelly@xyz.org'

In [220]: first_dict['id']
Out[220]: [23, 81]
```

Access with key

```
In [221]: first_dict.update({'id':[32,55]})
```

Modify dictionary: update

```
In [222]: first_dict
Out[222]: {'John': 'john@abc.com', 'Kelly': 'kelly@xyz.org', 'id': [32, 55]}

In [223]: del first_dict['id']
```

Modify dictionary: delete

```
In [224]: first_dict
Out[224]: {'John': 'john@abc.com', 'Kelly': 'kelly@xyz.org'}
```

Dictionary in Python

Problem Statement: You are instructed to print a dictionary with mixed keys and add elements. You are also asked to delete a key value.

Steps to Perform:

1. Create a dictionary with mixed keys
2. Add elements to the dictionary
3. Add a set of values to a single key
4. Access an element using a key
5. Remove elements from a dictionary
6. Delete a key from nested dictionary

ASSISTED PRACTICE

Dictionary and Its Operations

Problem Statement: After the client's review you have been asked to add multiple feedback actions on a nested dictionary. One of the action items is to create one dictionary that will contain the other three dictionaries and access the key-value pair. Also check if the key exists or not, then delete the dictionary and make a copy of the dictionary as well.

Steps to Perform:

1. Create a nested dictionary
2. Create three dictionaries, then create one dictionary that will contain the other three dictionaries
3. Access key-value pair using get() method
4. Update existing key's value
5. Check if key exists
6. Delete an entire dictionary
7. Make a copy of a dictionary using the copy() method

UNASSISTED PRACTICE

Unassisted Practice: Dictionary and Its Operations

```
[1]: Mydict = {
        "Employee1" : {
          "Name" : 'Chandler',
          "Joining_Date" : 1991
        },
         "Employee2" : {
           "Name" : 'Ross',
           "Joining_Date" : 1992
         },
         "Employee3" : {
           "Name" : 'Joey',
           "Joining_Date": 1993
         }
     }
     print(Mydict)
```

A dictionary containing three other dictionaries

```
{'Employee1': {'Name': 'Chandler', 'Joining_Date': 1991}, 'Employee2': {'Name': 'Ross', 'Joining_Date': 1992}, 'Employee3': {'Na
me': 'Joey', 'Joining_Date': 1993}}
```

Output

Unassisted Practice: Dictionary and Its Operations

```
[3]: House1 = {
       "name" : "Stark",
       "year" : 2001
     }
     House2 = {
       "name" : "Bolton",
       "year" : 2002
     }
     House3 = {
       "name" : "Lannister",
       "year" : 2003
     }

     GoT = {
       "House1" : House1,
       "House2" : House2,
       "House3" : House3
     }
     print(GoT)
```

Three separate dictionaries

One dictionary containing the other three dictionaries

```
{'House1': {'name': 'Stark', 'year': 2001}, 'House2': {'name': 'Bolton', 'year': 2002}, 'House3': {'name': 'Lannister', 'year':
2003}}
```

Output

Unassisted Practice: Dictionary and Its Operations

[5]:
```
 dict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
 }
print(dict.get("brand"))
```

Accessing key-value pair using get() method

```
Ford
```

[6]:
```
dict["model"] = "Mercury Cougar GT-E"
dict["year"] = 1968
print(dict)
```

Updating an existing key-value pair

```
{'brand': 'Ford', 'model': 'Mercury Cougar GT-E', 'year': 1968}
```

[7]:
```
if "brand" in dict:
    print("Brand exist:", dict["brand"])
```

Checking if a key exists

```
Brand exist: Ford
```

[8]:
```
dict1 = dict.copy()
print(dict1)
```

Making a copy of an existing dictionary

```
{'brand': 'Ford', 'model': 'Mercury Cougar GT-E', 'year': 1968}
```

[ ]:
```
del dict
```

Deleting an existing dictionary

Conditions and Branching

Conditional Statements

Control or conditional statements allow to check conditions and change the behavior of a program.

Abilities:

- Runs a selected section
- Controls the flow of a program
- Covers different scenarios

Example:

```
if x>0:
        print ("x is positive")
```

If Statement

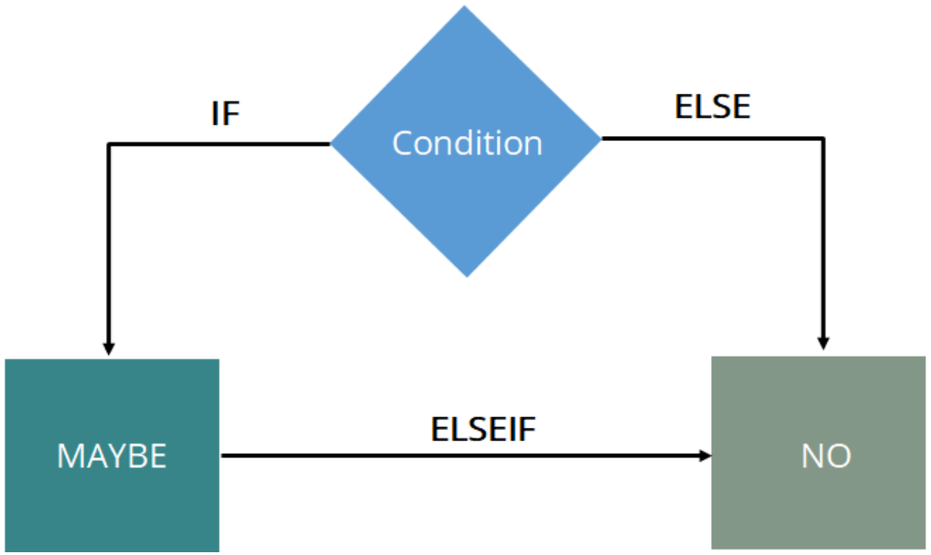
The if statement changes the flow of control in a Python program. This is used to run conditional statements.

If Condition = True	The code runs
If Condition = False	Nothing happens

Example:

if x > 0:

print ("x is positive")



If...Else Statements

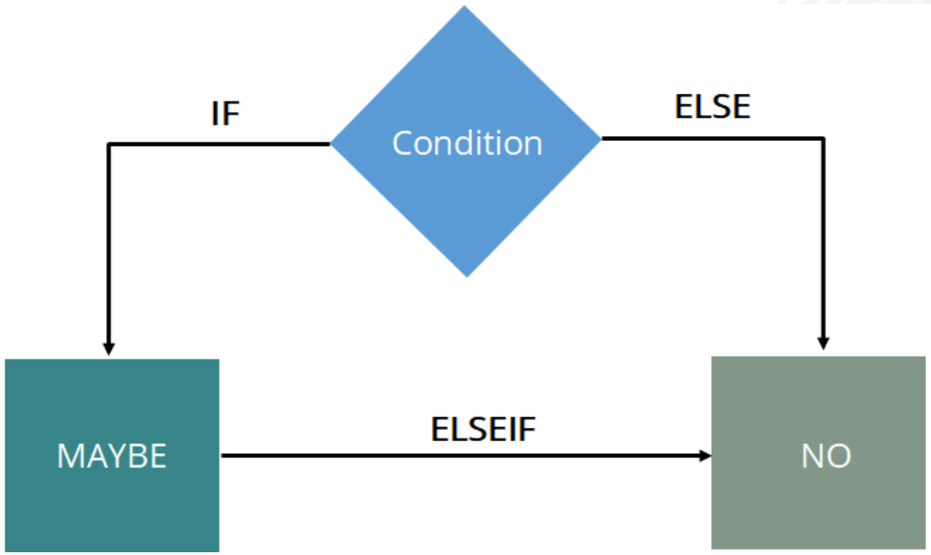
These are used to control the flow of a program and run conditional blocks.

Example 1

```
age = 20
if age == 20:
        print ("age is 20 years")
else:
        print ("age is not 20")
```

Example 2

```
if age > 18:
        print ("person is adult")
else:
        print("person is not adult")
```



If...Else If Statements

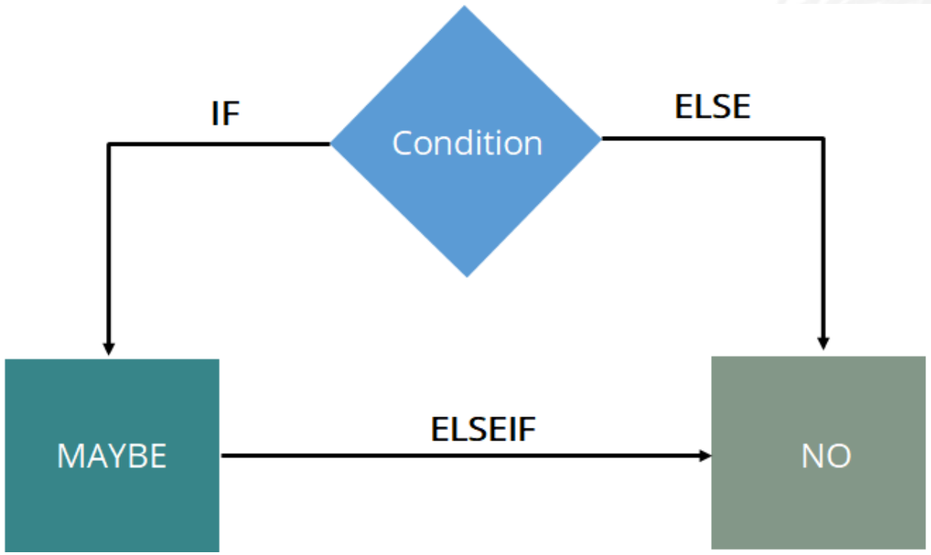
These are used to combine multiple if statements.

These statements:

- Execute only one branch
- Can be innumerable

Example:

```
marks= 95
If marks > 90:
        print ("A grade")
elif marks >= 80:
        print ("B grade")
elif marks >= 60
        print ("C grade")
```

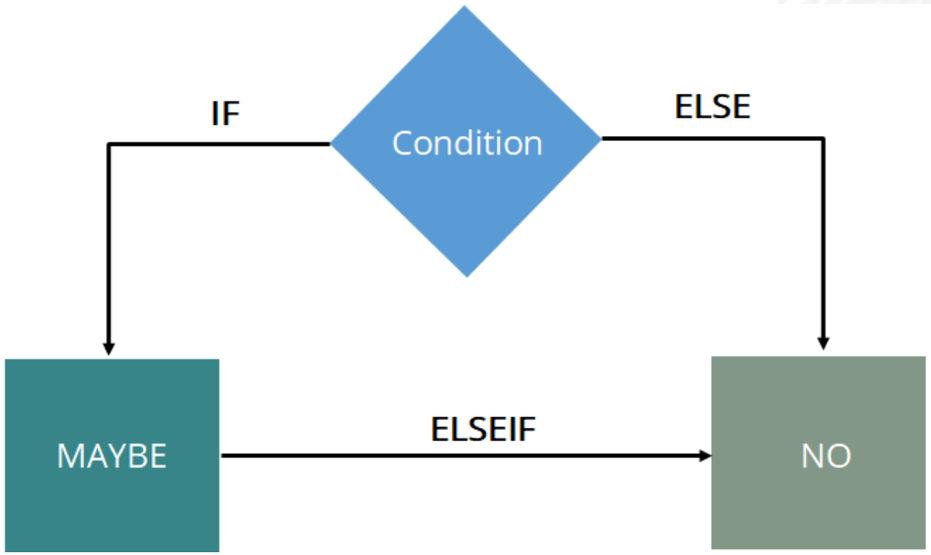


Elif Statements

These are combined statements. The *else* statement is executed when none of the conditions are met.

Example:

```
marks= 95
If marks > 90:
        print ("A grade")
elif marks >= 80:
        print ("B grade")
elif marks >= 60
        print ("C grade")
Else:
        print ("fail")
```



If, Elif, Else Statements: Example

The *if, elif,* and *else* statements are the most commonly used control flow statements.

```
[11]: age = 21
```

If condition

```
[12]: if age<20:
          print('minor')
      else:
          print('adult')

      adult
```

Else block

```
[13]: marks=81
```

```
[14]: if marks>90:
          print ('grade A')
      elif 80<=marks<=90:
          print ('grade B')
      elif 70<=marks<=80:
          print ('grade C')
      elif 60<=marks<=70:
          print ('grade D')
      else:
          print ('grade F')

      grade B
```

Nested if, elif, and else

Ternary Operators

Ternary operators, also known as conditional statements, test a condition in a single line, replacing the multiline if...else. This makes the code compact.

Syntax: [on_true] if [expression] else [on_false]

Example

```
max = (a>b) ? A : b
```

Check the Scores of a Course

Problem Statement : The school committee wants to automate course score calculation work. So they contacted the developing team to find a way for them. You have to select a course and set a criteria as per the school score rule and regulations.

Steps to Perform:

1. If the selected course is math
2. If the score in theory is more than 60 or score in practical is more than 50
3. Print the score of the subject
4. If the selected course is science
5. If the score in theory is more than 60 or score in practical is more than 40
6. Print the score of the subject

ASSISTED PRACTICE

While Loop

While Loop

The *while* statement is used for repeated execution if an expression is true.

Syntax :

```
<start value>
while  condition:
        statements
```

Example

```
a = 0
        while a < 3:
                a = a + 1
                print a
        print 'All Done'
```

While Loop: Example

Example code of a **while** loop

```
In [283]: temperature =   100
          while temperature > 95:
              print(temperature)
              temperature = temperature - 1
```

While condition

```
100
99
98
97
96
```

While Loop with Else

- *While* loop in Python can have an optional else part.
- The statements in the else part are executed when the condition is not fulfilled anymore.
- If the statements of the additional else part are placed right after the while loop without an else, they will be executed anyway.

Example

```
to_be_guessed = 5
guess = 0
while guess != to_be_guessed:
    guess = int(input("New number: "))
    if guess > 0:
        if guess > to_be_guessed:
            print("Number too large")
        elif guess < to_be_guessed:
            print("Number too small")
    else:
        print("Sorry that you're giving up!")
        break
else:
    print("Congratulation. You made it!")
```

Find Even Digit Numbers

Problem Statement: Write a program which will find all the numbers between 1000 and 3000 (both included) such that each digit of the number is an even number. The numbers obtained should be printed in a comma-separated sequence on a single line.

Steps to Perform:

1. Store an empty array to a variable
2. Check if it is not equal to 3001
3. Check if it is an even digit number
4. Print the values in a comma separated sequence

ASSISTED PRACTICE

Fibonacci Series Using While Loop

Problem Statement: Write a program to print Fibonacci series up to a certain number.

Steps to Perform:

1. Enter the last number of the series
2. Enter the first number as 0 and second as 1
3. Use a loop to find fibonacci series until fib_num (last number)
4. Print the fibonacci number

UNASSISTED PRACTICE

Unassisted Practice: Fibonacci Series Using While Loop

```
[6]: fib_num = 89
     first_num = 0
     second_num = 1

     while first_num <= fib_num:

         print(first_num)
         nth_num = first_num + second_num
         first_num = second_num
         second_num = nth_num
```

- `fib_num = 89` → Last number of the Fibonacci series
- `first_num = 0` → First number of a Fibonacci is zero
- `second_num = 1` → Second number in a Fibonacci series is one
- `while first_num <= fib_num:` → While loop to print the Fibonacci series until the 'fib_num'
- `print(first_num)` → Print the Fibonacci number
- `nth_num = first_num + second_num` → Print the Fibonacci number
- `first_num = second_num` / `second_num = nth_num` → Update the values of the first number and second number

```
0
1
1
2
3
5
8
13
21
34
55
89
```

Output

For Loop

For Loop

Python for loop is an iterator-based for loop. It steps through the items of lists, tuples, strings, keys of dictionaries, and other iterables.

Syntax :

```
for <variable> in <sequence>:
    <statements>
else:
    <statements>
```

Example

```
country=["India","USA","UK","China"]
for c in country:
    print(c)
```

Range Function

The range function in Python is usually used with the loop statements that provide a list of numbers, starting from zero to a number lesser than the given number.

Example

```
print(list(range(3,10)))
```

Output : [3, 4, 5, 6, 7, 8, 9]

Example

```
print(list(range(10)))
```

Output : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

For Loop with Range: Example

Example of *for loop* with range:

Range Example

List Example

```
for x in range(10):
        print (x)
```

For Loop with List: Example

Example of *for loop* with list:

Range Example

List Example

```
fruits = ['apple' , 'mango' , 'orange' , 'banana']
for fruit in fruits:
        print (fruit)
```

For Loop: Example

Example of a **for** loop

```
In [278]: stock_tickers =['AAPL','MSFT','GOOGL',None,'AMZN','CSCO','ORCL']
```

```
In [279]: for tickers in (stock_tickers):
              if(tickers is None):
                  continue
              print ('tickers')
```

For loop iterator

The **continue** statement

```
AAPL
MSFT
GOOGL
AMZN
CSCO
ORCL
```

```
In [280]: for tickers in (stock_tickers):
              if(tickers is None):
                  break
              print ('tickers')
```

The **break** statement

```
AAPL
MSFT
GOOGL
```

For Loop with Else

- It works exactly as the optional else of a while loop.
- It will be executed only if the loop hasn't been broken by a break statement.
- It will only be executed after all the items of the sequence in the header have been used.

Example

```
edibles = ["ham", "spam","eggs","nuts"]
for food in edibles:
    if food == "spam":
        print("No more spam please!")
        break
    print("Great, delicious " + food)
else:
    print("I am so glad: No spam!")
print("Finally, I finished stuffing myself")
```

Calculate the Number of Letters and Digits

Problem Statement: An intern was assigned a task by the manager to write a program that accepts a sentence and calculates the number of letters and digits.

Example: If the entered string is: Python0325

Then the output will be:

LETTERS: 6

DIGITS:4

Steps to Perform:

1. Enter a string
2. Set the value of 0 for letters and digits
3. Calculate the numbers of letters and digits
4. Print the number of letters and digits

ASSISTED PRACTICE

Create a Pyramid of Stars

Problem Statement: Write a program to print a pyramid of stars.

Example:

```
    *
   * * *
  * * * * *
 * * * * * * *
* * * * * * * * *
```

Steps to Perform:

1. Enter the number of rows
2. Use double loop to print rows and number of spaces
3. Use a loop to print odd number of stars
4. Print stars and append a space

UNASSISTED PRACTICE

Unassisted Practice: Create a Pyramid of Stars

```
[2]: rows = 6
k = 0
for i in range(1, rows+1):
    for space in range(1, (rows-i)+1):
        print(end="  ")
    while k != (2*i-1):
        print("* ", end="")
        k = k + 1
    k = 0
    print()
```

Number of rows in the pyramid

Loop for the rows

Loop to print the number of spaces

A **while** loop to print the odd number of stars in each row

Print the stars and append a space

New line after each row to display pattern correctly

```
            *
          * * *
        * * * * *
      * * * * * * *
    * * * * * * * * *
  * * * * * * * * * * *
```

Output

Break and Continue Statements

Break Statement

The break statement breaks out of the innermost enclosing for or while loop.
The break statement may only occur syntactically nested in a for or while loop.

Example

```
counter = 1
while counter <= 10:
    if(counter % 2 ==0 and counter %3 == 0):
                    break
    print(counter)
    counter = counter+1
```

Continue Statement

The continue statement may only occur syntactically nested in a for or while loop.
It continues with the next cycle of the nearest enclosing loop.

Example

```
counter = 1
while counter <= 10:
    if(counter % 5 ==0):
                    continue
    print(counter)
    counter = counter+1
```

Knowledge Check

Knowledge Check 1	**Name the programming model that consists of objects.**

a. Structured programming

b. Aspect-oriented programming

c. Service-oriented architecture

d. Object-oriented programming

Knowledge Check 1	**Name the programming model that consists of objects.**

a. Structured programming

b. Aspect-oriented programming

c. Service-oriented architecture

d. Object-oriented programming

The correct answer is **d**

Object-oriented programming revolves around objects and the data and behavior associated with them.

Knowledge Check 2	What is used to mark a block of code in Python?

a. Curly braces

b. Square brackets

c. Indentation

d. Semicolon

Knowledge Check 2	What is used to mark a block of code in Python?

a. Curly braces

b. Square brackets

c. Indentation

d. Semicolon

The correct answer is **c**

Indentation marks the block of code.

Knowledge Check 3	Name the statement that exits the control from a function in Python.

a. Break

b. Exit

c. Return

d. Back

Knowledge Check 3	**Name the statement that exits the control from a function in Python.**

a. Break

b. Exit

c. Return

d. Back

The correct answer is **c**

The return statement helps exit the control from a function.

Knowledge Check 4	When is “*args” used?

a. To create a recursive function

b. To pass arbitrary number of arguments to a function

c. To set the default value of a function argument

d. To reuse code

Knowledge Check 4	When is “*args” used?

a. To create a recursive function

b. To pass arbitrary number of arguments to a function

c. To set the default value of a function argument

d. To reuse code

The correct answer is **b**

“*args” is used to pass arbitrary number of arguments to a function.

Key Takeaways

- A variable can be assigned or bound to any value.
- Python also supports the Null and Boolean data types.
- Function overloading happens when you have more than one function with the same name.
- Python data structure consists of tuples, lists, sets, and dictionaries.
- Loops in Python consist of while, for, continue, and break.

Tic-Tac-Toe Game

Duration: 30 mins

Problem Statement:

Write a program logic to check whether someone has won a game of tic-tac-toe, without considering the moves.

LESSON-END PROJECT