Unit I Basics Of Python

Python programming language: About Python- Introduction to various IDEs- IDLE- PyCharm, Spyder- Sublime text- Jupyter Notebook.
Literals: Numeric literals - String literals- Variables and Identifiers: Variable assignment and keyboard input – Identifiers - keywords and other predefined identifiers.
Control Structures: Sequential control- Selection control- Iterative control statements

1.1 The Python Programming Language

1.1.1 About Python

- Guido van Rossum is the creator of the Python programming language, first released in the early 1990s.

Features / Characteristics of Python

- Python has a simple syntax.
- Python programs are clear and easy to read.
- Python provides powerful programming features.
- Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh
- Companies and organizations that use Python include YouTube, Google, Yahoo, and NASA.
- Python is Interpreted
- Python is Object-Oriented
- Python is designed to be highly readable.
- Python can be used on a server to create web applications.
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- It supports functional and structured programming methods as well as OOP.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Uses of python

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

1.1.2 IDE

- IDE (Integrated Development Environment)
- An IDE is a bundled set of software tools for program development.
- IDEs are full-fledged environment which provide all the essential tools needed for software development.
- It just doesn't handle the code (for example, write, edit, syntax highlighting and auto-completion) but also provides other features such as debugging, execution, testing, and code formatting that helps programmers.
- It understands your code much better than a text editor.
- It usually provides features such as build automation, code writing, testing and debugging. This can significantly speed up your work. The downside is that IDEs can be complicated to use.

Types of IDEs

- ✓ IDLE
- ✓ Pycharm
- ✓ Jupyter
- ✓ Spyder
- ✓ Sublime Text

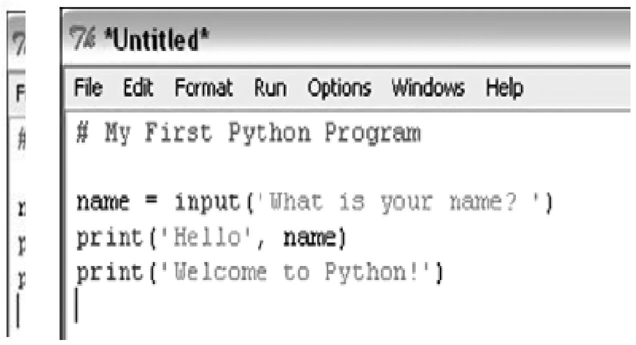
IDLE

- IDLE (Integrated Development and Learning Environment).
- It includes an **editor** for creating and modifying programs
- A **translator** for executing programs
- **Program debugger** to aid in finding program errors.
- Python provides the very useful ability to execute in interactive mode.
- The window that provides this interaction is referered to as the **Python shell** .
- Working in the Python shell is convenient, the entered code is not saved in shell.
- Interacting with the shell is much like using a calculator, except that, instead of being limited to the operations built into a calculator (addition, subtraction, etc)

- It is a default editor that accompanies Python

- This IDLE is suitable for beginner level developers

- The IDLE tool can be used on Mac OS, Windows, and Linux
- **Features of IDLE:**
    - ✓ Ability to search for multiple files
    - ✓ Interactive interpreter with syntax highlighting, and error and i/o messages
    - ✓ Smart indenting, along with basic text editor features
    - ✓ A very capable debugger
- First, to create a Python program file, select New Window from the File menu in the Python shell.
- Type the following in the program window exactly as shown



```
# My First Python Program

name = input('What is your name? ')
print('Hello', name)
print('Welcome to Python!')
```

- Next save the program file by selecting Save As under the File menu, and save in the appropriate folder with the name MyFirstProgram.py. and execute it by pressing F5.

Advantage:

- It can be used to execute a single statement.
- It can be used to create, modify, and execute Python scripts.
- It offers features like syntax highlighting, auto-completion & smart indent.
- It has a debugger with stepping & breakpoint features.

Disadvantage:

- IDLE is not available by default in python distribution for Linux.
- It needs a respective package manager for installation.

PyCharm

- PyCharm is a widely used Python IDE created by JetBrains
- It has been considered the best IDE for python developers.

- This IDE is suitable for professional developers and facilitates the development of large Python projects
- **Features of PyCharm:**
    - ✓ Support for JavaScript, CSS, and TypeScript
    - ✓ Smart code navigation
    - ✓ Quick and safe code refactoring
    - ✓ Support features like accessing databases directly from the IDE
    - ✓ It is considered as an intelligent code editor, fast and safe refactoring, and smart code.
    - ✓ Features for debugging, profiling, remote development, testing the code, auto code completion, quick fixing, error detection and tools of the database.
    - ✓ Support for Popular web technologies, web frameworks, scientific libraries and version control.

Advantage:

- Active community support
- Live code verification and syntax highlighting
- Executes edits and debugs Python code without any external requirements

Disadvantage:

- Slow loading time
- The default setting may require adjustment before existing projects can be used.

<u>Jupyter Notebook</u>

- Jupyter is widely used in the field of data science
- It is easy to use, interactive and allows live code sharing and visualization
- It is easy to use, interactive data science IDE across many programming languages
- It is not work as an editor, but also as an educational tool or presentation.
- **Features of Jupyter:**

- ✓ Supports for the numerical calculations and machine learning workflow
- ✓ Combine code, text, and images for greater user experience
- ✓ Inter generation of data science libraries like NumPy, Pandas, and Matplotlib
- ✓ It is one of the best Python IDE that supports for Numerical simulation, data cleaning machine learning data visualization, and statistical modeling.
- ✓ Combine code, text, and images.
- ✓ Support for many programming languages.
- ✓ Integrated data science libraries (matplotlib, NumPy, Pandas).

Advantage:

- It combines code, text, images, videos, mathematical equations, plots, maps, graphical user interface and widgets to a single document.
- It allows users to convert the notebooks into other formats such as HTML and PDF.
- It saved in the structured text files (JSON format), which makes them easily shareable.
- It is platform-independent because it is represented as JSON (JavaScript Object Notation) format, which is a language-independent, text-based file format.

Disadvantage:

- It is very hard to test long asynchronous tasks.
- Less Security
- It runs cell out of order
- In Jupyter notebook, there is no IDE integration, no linting, and no code-style correction.

<u>Spyder</u>

- Spyder is an open-source IDE most commonly used for scientific development
- It is also called Scientific Python Development IDE and it is the most lightweight IDE for Python
- Spyder comes with Anaconda distribution, which is popular for data science and machine learning.
- It allows you to access PostgreSQL, Oracle, MySQL, SQL Server, and many other databases from the IDE.
- **Features of Spyder:**

- ✓ Support for automatic code completion and splitting
- ✓ Supports plotting different types of charts and data manipulation
- ✓ Integration of data science libraries like NumPy, Pandas, and Matplotlib
- ✓ Auto code completion and syntax highlighting
- ✓ An interactive way to trace each step of Python code execution
- ✓ It is very efficient in tracing each step of the script execution by a powerful debugger
- ✓ It offers automatic code completion and horizontal/vertical splitting

Advantage:

- Community support
- Rich in development tool features
- Complete documentation

Disadvantage:

- Execution dependencies
- Optional dependencies

Sublime Text

- Sublime Text is a generic text editor coded in C++ and Python.
- This software supports 44 major programming languages, including Python.
- It was first published in 2007, and ***Jon Skinner*** developed it.
- It is one of the best Python editor that has basic built-in support for Python.
- The editor supports OS X, Windows, and Linux operating systems
- Sublime text is used to create a full-fledged Python development environment
- **Features of Sublime Text:**
    - ✓ Discreet, minimal interface: we must be able to focus on the text and not a myriad of toolbars;
    - ✓ The text is not hidden by the windows;
    - ✓ Use as much space as possible: full screen, multi-screen, side-by-side file editing should be possible.
    - ✓ It supports different plugins and packages.

- ✓ It is high quality and powerful IDE.
- ✓ It incorporates most of the features of a basic Python text editor, including customizable syntax highlighting.

Advantage:

- Fast with very few bugs (big advantage)
- Opens large files
- Supports many languages

Disadvantage:

- Difficult to modify, everything goes through JSON.
- License required
- Learning the shortcuts

1.2 Literals

Python Literals

- Python Literals can be defined as data that is given in a variable or constant.
- literals are a notation for representing a fixed value in source code.
- Python supports the following literals:
  - ✓ Numeric literals
  - ✓ String literals
  - ✓ Boolean literals
  - ✓ Special literals

1.2.1 Numeric literals

- A **numeric literal** is a literal containing only the digits 0–9, an optional sign character and a possible decimal point. (The letter *e* is also used in exponential notation).
- If a numeric literal contains a decimal point, then it denotes a **floating-point value** , or " **float** " (e.g., 10.24); otherwise, it denotes an **integer value** (e.g., 10).
- They are immutable and there are three types of numeric literal :
  - ✓ **Integer**
  - ✓ **Float**

- ✓ **Complex.**

- **Integer :**
  - ✓ Both positive and negative numbers including 0. There should not be any fractional part.
  - ✓ There is no limit to the size of an integer that can be represented in Python
- **Float**
  - ✓ These are real numbers having both integer and fractional parts.
  - ✓ Floating-point values, however, have both a limited *range* and a limited *precision* . Python uses a double-precision standard format providing a range of 10^{-308} to 10^{308} with 16 to 17 digits of precision.
- **Complex Literal**
  - ✓ The numerals will be in the form of **a+bj**, where '**a**' is the real part and '**b**' is the complex part.
- Example:

```
a = 0b1010 #Binary Literals
b = 100 #Decimal Literal
c = 0o310 #Octal Literal
d = 0x12c #Hexadecimal Literal

#Float Literal
float_1 = 10.5
float_2 = 1.5e2

#Complex Literal
x = 3.14j

print(a, b, c, d)
print(float_1, float_2)
print(x, x.imag, x.real)
```

<u>Output</u>

```
10 100 200 300
10.5 150.0
3.14j 3.14 0.0
```

- **Built-in format Function**

  - ✓ The built-in **format** function can be used to produce a numeric string of a given floating-point value *rounded* to a specific number of decimal places.,

```
>>>12/5
  2.4
>>>format(12/5, '.2f')
```

```
>>>5/7
  0.7142857142857143
>>>format(5/7, '.2f')
```

```
  '2.40'                                        '0.71'
>>>format(13402.25, ' , .2f')
            13,402.24
```

1.2.2 String literals

- A string literal is a sequence of characters surrounded by quotes.
- A string literal can be created by writing a text(a group of Characters) surrounded by the single(”), double(“”), or triple quotes.
- By using triple quotes we can write multi-line strings.
- 'hello' is the same as "hello".
- Example:

```
s = 'hello'

t = " hello "

m = ''' hello
       world'''

print(s)
print(t)
print(m)
```

- Character literal is also a type of string literals where a single character surrounded by single or double-quotes.
- Example:

```
a = 'h'

b = "h"

print(a)
print(b)
```

- **The Representation of Character Values**

  ✓ There needs to be a way to encode (represent) characters within a computer.

  ✓ the **Unicode** encoding scheme is intended to be a universal encoding scheme utilizing between 8 and 32 bits for each character.

  ✓ The default encoding in Python uses **UTF-8**
  ✓ Unicode is capable of defining more than 4 billion characters.
  ✓ For example,
  'A' is encoded as 01000001 (65),

'B' is encoded as 01000010 (66)
'0' is encoded as 00110000 (48)
'1' is encoded as 00110001 (49).

- ✓ The ord func gives the UTF-8 (ASCII) encoding of a given char.
  ord('A') is 65.
- ✓ The chr function gives the character for a given encoding value.
  chr(65) is 'A'.

- **String Formatting**

  - ✓ The format function can be used to control how strings are displayed.
    - ▪ format(*value, format_specifi er)*
  - ✓ where *value* is the value to be displayed, and *format_specifi er* can contain formatting options.
  - ✓ For example:

```
format('Hello', ' < 20')    #→ 'Hello     '(Left-justified in a field width of 20 char)
format('Hello', ' > 20')    #→ '     Hello'(Right-justified in a field width of 20 char)
format('Hello', '^20')    #→ '  Hello   '(Center-justified in a field width of 20 char)
format(' ', '30')                  #→ '          ' (strings of 30 blank characters,)
```

1.2.3 Variables and Identifies

Python Variables

- A variable is a named location used to store data in the memory.
- A **variable** is a name that is associated with a value.
- Variables are containers for storing data values.
- variable names are case-sensitive
- For example,

```
number = 10
```

- created a variable named *number*. We have assigned the value 10 to the variable.

```
number = 10
number = 1.1
```

- Initially, the value of *number* was 10. Later, it was changed to 1.1.

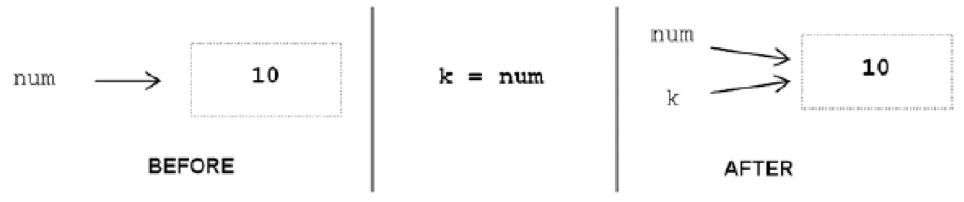


FIGURE 2-9 Variable Assignment (to another variable)

```
>>>id(num)          >>>id(k)
505494040           505494040
```

- The id function produces a unique number identifying a specific value (object) in memory.
- The same variable can be associated with values of different type during program execution, as indicated below.

```
var = 12          integer
var = 12.45       float
var = 'Hello'     string
```

Variable Assignment and Keyboard Input

```
>>>name = input('What is your first name?')
What is your first name? Cathy
```

- All input is returned by the input function as a string type.
- *For the input of numeric values, the response must be converted to the appropriate type.*
- Python provides built-in **type conversion functions int** () and **float** ().

```
a = int(input('Enter a Number :'))
b = float(input('Enter average :'))
```

Assigning values to Variables in Python

- Assignment operator (=) is used to assign a value to a variable.
- Example:

```
website = "apple.com"
print(website)
```

Output

```
apple.com
```

- In the above program, we assigned a value apple.com to the variable *website*.

Changing the value of a variable

```
website = "apple.com"
print(website)

# assigning a new value to website
website = "program.com"

print(website)
```

Output

```
apple.com
program.com
```

Assigning multiple values to multiple variables

```
a, b, c = 5, 3.2, "Hello"

print (a)
print (b)
print (c)
```

- If we want to assign the same value to multiple variables at once, we can do this as:

```
x = y = z = "hai"

print (x)
print (y)
print (z)
```

Rules and Naming Convention for Variables and constants

1. Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (**A to Z**) or digits (**0 to 9**) or an underscore (**_**).
2. Create a name that makes sense.
3. If you want to create a variable name having two words, use underscore to separate them. For example: my_name
4. Use capital letters possible to declare a constant. For example: PI, MASS,TEMP
5. Never use special symbols like !, @, #, $, %, etc.
6. Don't start a variable name with a digit.
7. Variable names are case-sensitive (age, Age and AGE are three different variables)

<u>Identifier</u>

- An **identifier** is a sequence of one or more characters used to provide a name for a given program element.

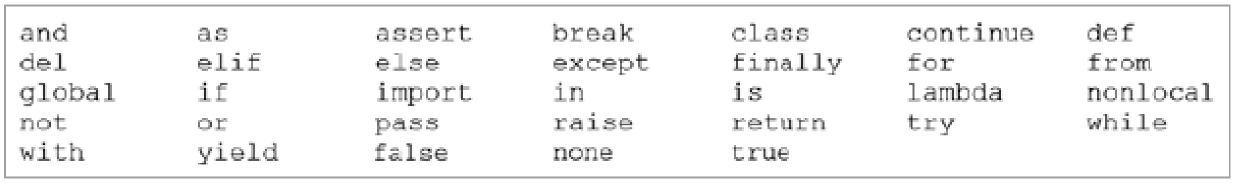
- A Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python is *case sensitive* , thus, Line is different from line.
- Identifiers may contain letters and digits, but cannot begin with a digit.
- The underscore character, _, is also allowed, but it should not be used as the *first* character, however, as identifiers beginning with an underscore have special meaning in Python.
- Spaces are not allowed as part of an identifier.

Keywords and Other Predefined Identifiers in Python

- Keywords are reserved words
- A **keyword** is an identifier that has predefined meaning in a programming language.
- All the Python keywords contain lowercase letters only
- keywords cannot be used as “regular” identifiers.

```
>>>and =10
SyntaxError: invalid syntax
```

To display the keywords, type help() in the Python shell, and then type keywords (type 'q' to quit).



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

FIGURE 2-12 Keywords in Python

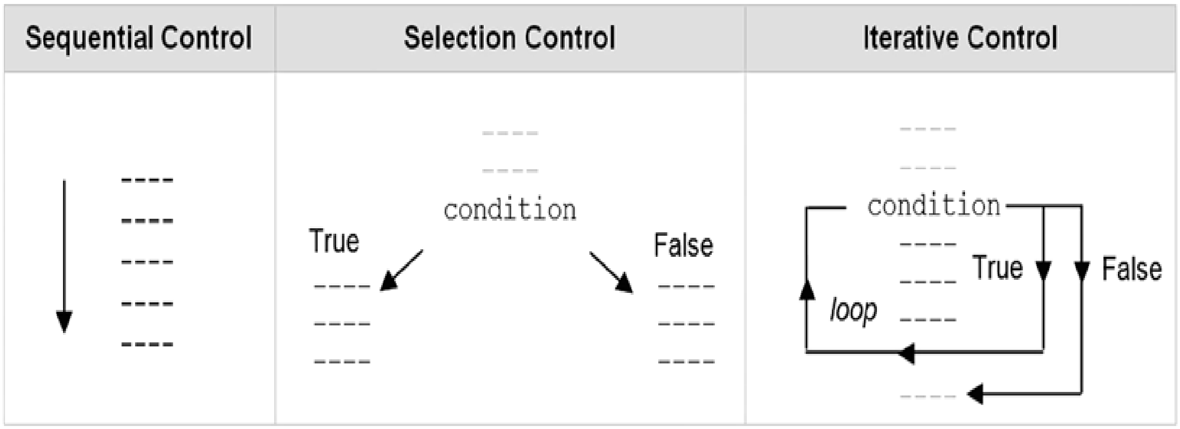
- A simple way to check whether a given identifi er is a keyword in Python is given below,

```
>>>'exit' in dir(__builtins__)
True
```

1.3 Control Structures

- C*ontrol flow* is the order that instructions are executed in a program.
- A **control statement** is a statement that determines the control flow of a set of instructions.
- A **control structure** is a set of instructions and the control statements controlling their execution.
- There are three fundamental forms of control that programming languages provide
  - ✓ *sequential control*
  - ✓ *selection control*
  - ✓ *iterative control*

- **Sequential control** is an implicit form of control in which instructions are executed in the order that they are written.
- A program consisting of only sequential control is referred to as a “straight-line program.”
- **Selection control** is provided by a control statement that *selectively executes* instructions
- **Iterative control** is provided by an iterative control statement that *repeatedly executes* instructions.

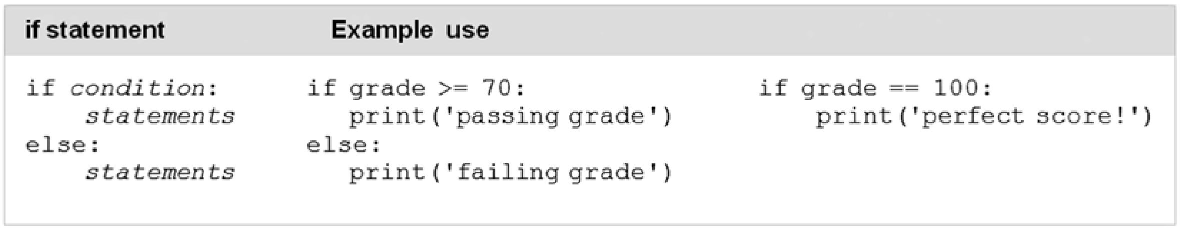


1.3.1 Selection Control

- A **selection control statement** is a control statement providing selective execution of instructions.
- A *selection control structure* is a given set of instructions and the selection control statement(s) controlling their execution

If Statement

- An **if statement** is a selection control statement based on the value of a given Boolean expression.
-



if statement	Example use	

`if condition:`
`    statements`
`else:`
`    statements`
	`if grade >= 70:`
	`   print('passing grade')`
	`else:`
	`   print('failing grade')`
		`if grade == 100:`
		`    print('perfect score!')`

- Statements that contain other statements are referred to as a **compound statement**.

✓ Example

a=int(input('Enter a Number'))

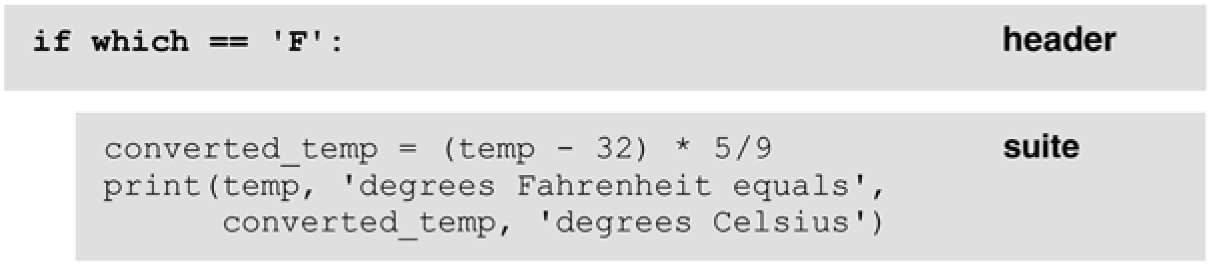
```
if a%2==0:
   print("It is Even Number")
else:
   print("It is odd Number")
```

✓ Output

```
Enter a Number78
It is Even Number
```

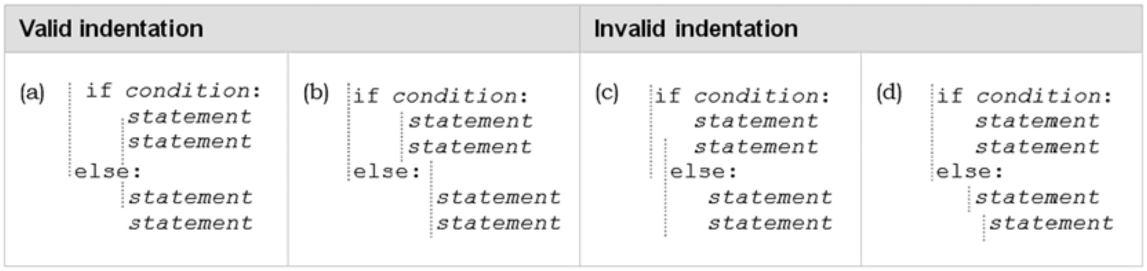
Indentation in Python

- Indentation is simply used to align program lines to aid readability.
- In Python, however, indentation is used to associate and group statements



```
if which == 'F':
    converted_temp = (temp - 32) * 5/9
    print(temp, 'degrees Fahrenheit equals',
          converted_temp, 'degrees Celsius')
```

- The set of statements following a header in Python is called a **suite** (commonly called a **block**).
- The statements of a given suite must all be indented the same amount.
- A header and its associated suite are together referred to as a **clause** .



Valid indentation

(a)
```
if condition:
    statement
    statement
else:
    statement
    statement
```

(b)
```
if condition:
        statement
        statement
else:
            statement
            statement
```

Invalid indentation

(c)
```
if condition:
    statement
    statement
  else:
      statement
      statement
```

(d)
```
if condition:
    statement
    statement
else:
    statement
     statement
```

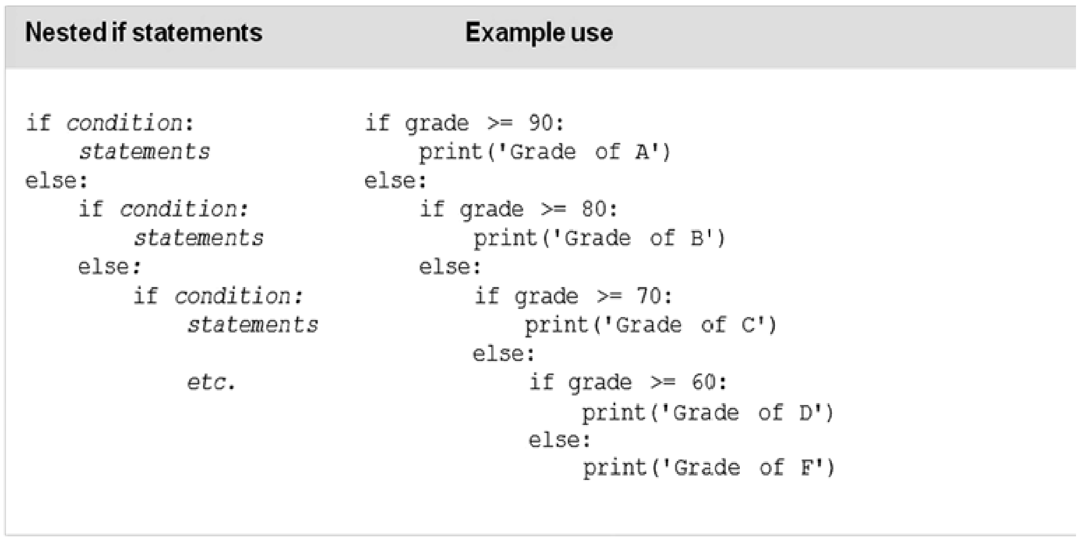
<u>Multi-Way Selection</u>

Two ways of constructing multi-way selection in Python:

1. nested if statements
2. elif headers.

Nested if Statements

- When selection among more than two sets of statements (suites) is needed.
- For such situations, if statements can be nested, resulting in **multi-way selection.**



Nested if statements	Example use

```
if condition:              if grade >= 90:
    statements                 print('Grade of A')
else:                      else:
    if condition:              if grade >= 80:
        statements                 print('Grade of B')
    else:                      else:
        if condition:              if grade >= 70:
            statements                 print('Grade of C')
                                   else:
            etc.                       if grade >= 60:
                                           print('Grade of D')
                                       else:
                                           print('Grade of F')
```

✓ Example

```
var = 100
if var < 200:
   if var == 150:
     print ("Which is 150")
  elif var == 100:
     print ("Which is 100")
  elif var == 50:
     print ("Which is 50")
  elif var < 50:
     print ("Value is less than 50")
else:
  print ("Values more than 200..")
```

- In the first if statement, if variable grade is greater than or equal to 90, then 'Grade of A' is displayed.
- Therefore, its else suite is not executed, containing the remaining if statements.
- If grade is less than 90, the else suite is executed.
- If grade is greater than or equal to 80, 'Grade of B' is displayed and the rest of the if statements in *its* else suite are skipped, and so on.
- The final else clause is executed only if all the previous conditions fail, displaying 'Grade of F'. This is referred to as a *catch-all* case.

elif Header

- elif ("else-if") that provides multi-way selection in a single if statement,

- The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".
- An **else** statement can be combined with an **if** statement.
- An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

Syntax:	Example:

```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
```
	```
	a=int(input('Enter a Number'))
	if a>0:
	    print(a," is Positive Number")
	elif a<0:
	    print(a," is Negative Number")
	else:
	    print(a," is Zero")
	```
	 ✓ **Output**
	Enter a Number-67
	-67 is Negative Number

1.3.2 Iterative Control Statements/looping control statements

- An **iterative control statement** is a control statement providing the repeated execution of a set of instructions.
- An *iterative control structure* is a set of instructions and the iterative control statement(s) controlling their execution.
- Because of their repeated execution, iterative control structures are commonly referred to as "loops."

While Statement

- A **while statement** is an iterative control statement that repeatedly executes a set of statements based on a provided Boolean expression (condition).
- If the condition of a while statement is true, the statements within the loop are (re)executed.
- Once the condition becomes false, the iteration terminates and control continues with the fi rst statement after the while loop.
- The first time a loop is reached, the condition may be false, and therefore the loop would never be executed.

• Syntax	• Example

```
while condition:
        Statements
```
	```
	s=0
	i=1
	n=int(input("Enter a number :"))
	while i<=n:
	        s=s+i
	```

	i=i+1
	print(s)

- Example:

```
#print 1 to 5
i = 1
while i < 6:
    print(i)
    i=i+1
```

✓ Output:

```
1
2
3
4
5
```

```
#print even number until 10
j=1
print("Even Numbers are")
while j<=10:
    if j%2==0:
        print(j)
    j=j+1
```

✓ Output:

```
Even Numbers are
2
4
6
8
10
```

For

✓ The for **loop in Python** is used to iterate the statements or a part of the program several times.

✓ It is frequently used to traverse the data structures like list, tuple, or dictionary.

✓ syntax

for iterating_var **in** sequence:
  statement(s)

✓ Example

```
#print natural numbers
i=1
n=int(input("Enter the number up to print the natural numbers : "))
for i in range(0,n):
    print(i,end=' ')
```

- ✓ Output

```
Enter the number up to print the natural numbers : 5
0 1 2 3 4
```

- ✓ Example

```
#print String
s="Hello World"
print("\n")
for c in s:
    print(c,end=' ')
```

- ✓ Output

```
H e l l o   W o r l d
```

- ✓ Example

```
#Multiplication Table
i=1;
print("\n")
num = int(input("Enter a number:"));
for i in range(1,11):
    print("%d X %d = %d"%(num,i,num*i));
```

- ✓ Output

```
Enter a number:4
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
```

range() Function

- ✓ To loop through a set of code a specified number of times, we can use the range() function,
- ✓ The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
- ✓ The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):
- ✓ Program

```
for x in range(6):
     print(x)

for x in range(2, 6):
  print(x)

for x in range(2, 30, 3):
  print(x)
```

- ✓ Output

```
0 1 2 3 4 5
2 3 4 5
2 5 8 11 14 17 20 23 26 29
```