CST-362 PROGRAMMING IN PYTHON

Syllabus - Module I

Programming Environment and Python Basics:
Getting started with Python programming – Interactive shell, IDLE, iPython Notebooks, Detecting and correcting syntax errors, How Python works. The software development process – A case study.

Basic coding skills – strings, assignment, and comments, Numeric data types and character sets, Expressions, Using inbuilt functions and modules. Control statements – Iteration with for/while loop, Formatting text for output, A case study, Selection structure (if-else, switchcase), Conditional iteration with while, A case study, Testing control statements, Lazy evaluation.

Introduction to Python

- Guido Van Rossum invented the Python programming language in the early 1990s.
- high-level, interpreted, interactive and object-oriented scripting language.

  ✓ Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.

  ✓ Python is Interactive – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

  ✓ Python is Object-Oriented – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Characteristics of Python

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code forA building large applications.
- 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.

Characteristics of Python

Readability and ease-of-maintenance

- Python focuses on well-structured easy to read code
- Easier to understand source code
- hence easier to maintain code base

- Portability
  - Scripting language hence easily portable
  - Python interpreter is supported on most modern OS's

- Extensibility with libraries
  - Large base of third-party libraries that greatly extend functionality. Eg., NumPy, SciPy etc.

Why Python?

- 1. Easy to learn:- It has few keywords, simple structure and clearly defined syntax.
- 2. Easy to read:- It is clearly defined
- 3. Easy to maintain:- It is fairly easy to maintain.
- 4. Portable:- It can run wide variety of hardware platforms and has the same interface on all platforms.
- 5. Databases:- It provides interfaces to all major commercial databases.
- 6. Extendable:- Adding of low level modules to python is possible.
- 7. Scalable:- It provides a better structure and support for larger programs.
- 8. It supports functional and structured programming.
- 9. It can be used as scripting language and can be compiled to byte code to form larger applications.
- 10. It supports automatic garbage collection.
- 11. It can be easily integrated with other programming languages like c,c++,java

The language is used by companies in real revenue generating products, such as:

- In operations of Google search engine, youtube, etc.
- Bit Torrent peer to peer file sharing is written using Python
- Intel, Cisco, HP, IBM, etc use Python for hardware testing.
- Maya provides a Python scripting API
- i–Robot uses Python to develop commercial Robot.
- NASA and others use Python for their scientific programming task.

versions

- The Python programming language was conceived in the late 1980s as a successor to the ABC language and its implementation was started in December 1989
- Python 2.0 was released on October 16, 2000
  Python 3.0, was released on December 3, 2008
  Python 3.7 and 2.7 are the two stable releases.
  The Python 2 was officially discontinued in 2020.
  Python 3.8 released in oct 14 2019Python 3.9 released in Oct 5 2020
- Python 3.10 is the latest stable release, released in Oct 4 2021
  Python 3.x is strongly recommended for any new development.

Python Installation

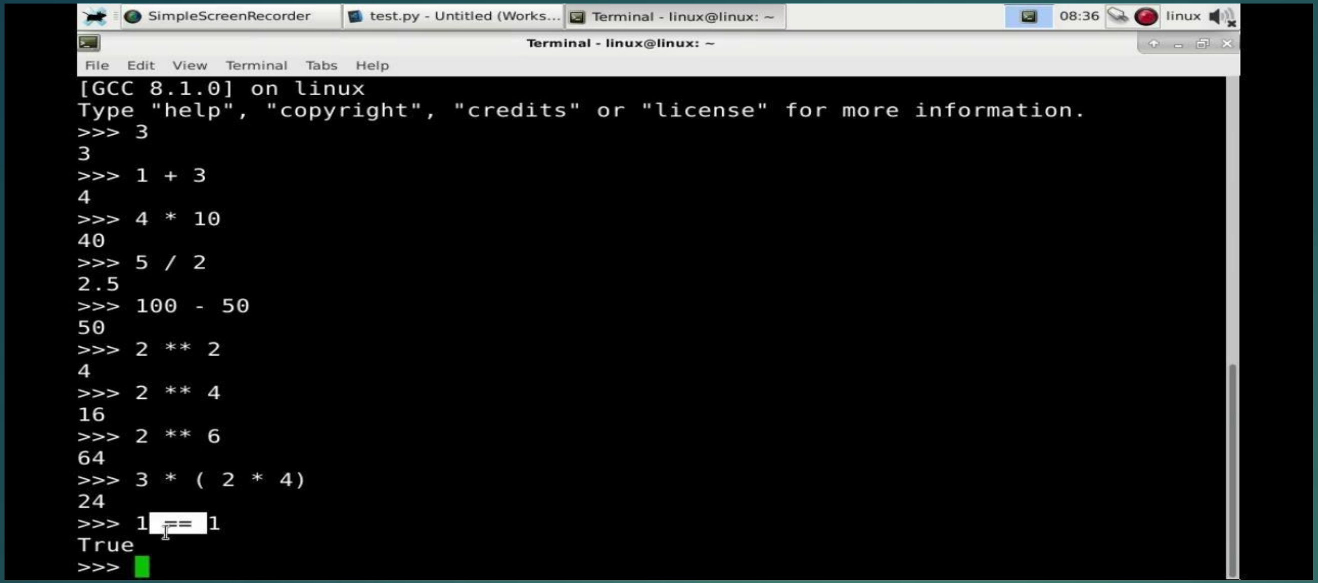
- • Python Install –
- To check if you have python installed on a Windows PC, search in th e start bar
- To check if you have python installed on a Linux or Mac, then on lin ux open the command line or on Mac open the Terminal and type: python --version
-

  If you find that you do not have python installed on your computer, t hen visit https://www.python.org/

INTERACTIVE PYTHON SHELL

- The Python interactive console, also known as Python interpreter or Python shell, provides programmers with a quick way to execute commands and try out and test code without creating a file.

- If you want to run programs stored in a file on this shell, type in python FILENAME.py where FILENAME is the name of your file. This is useful for debugging.



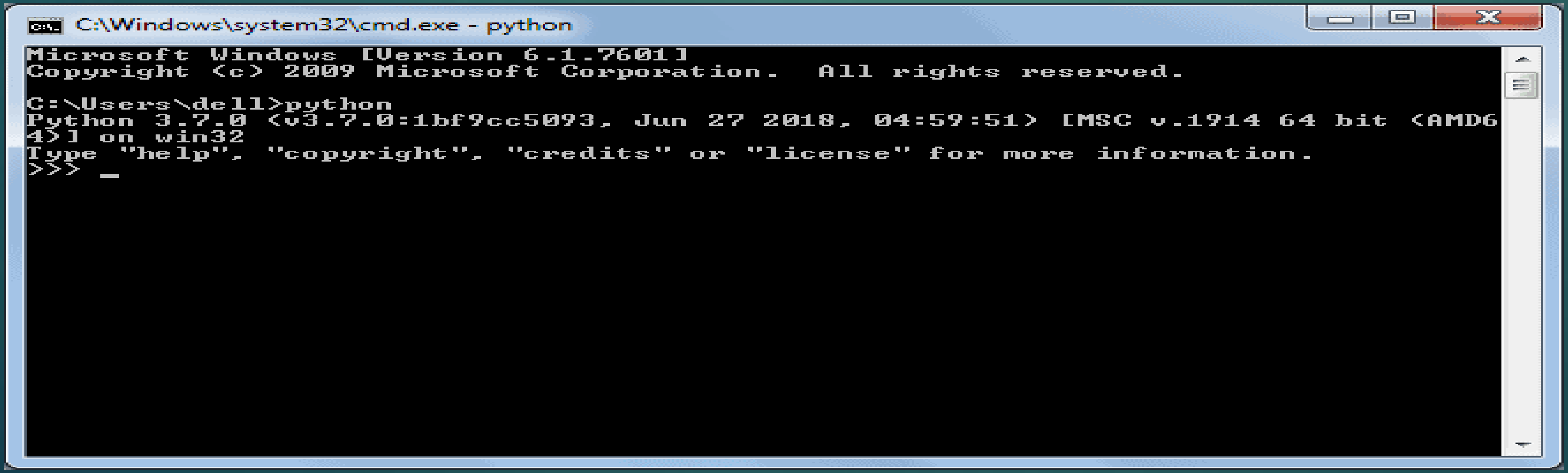
SimpleScreenRecorder | test.py - Untitled (Works... | Terminal - linux@linux: ~ | 08:36 | linux

Terminal - linux@linux: ~

File Edit View Terminal Tabs Help

```
[GCC 8.1.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 3
3
>>> 1 + 3
4
>>> 4 * 10
40
>>> 5 / 2
2.5
>>> 100 - 50
50
>>> 2 ** 2
4
>>> 2 ** 4
16
>>> 2 ** 6
64
>>> 3 * ( 2 * 4)
24
>>> 1 == 1
True
>>>
```

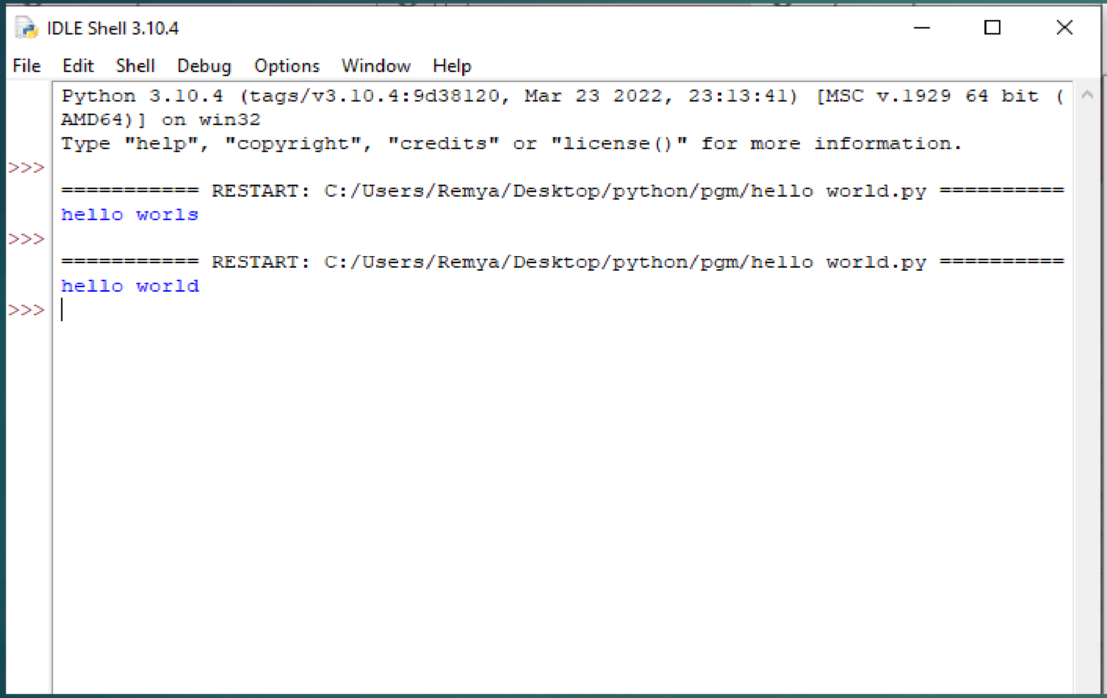
The Interactive Shell can execute Python commands, Logical commands, arithmetic commands etc.



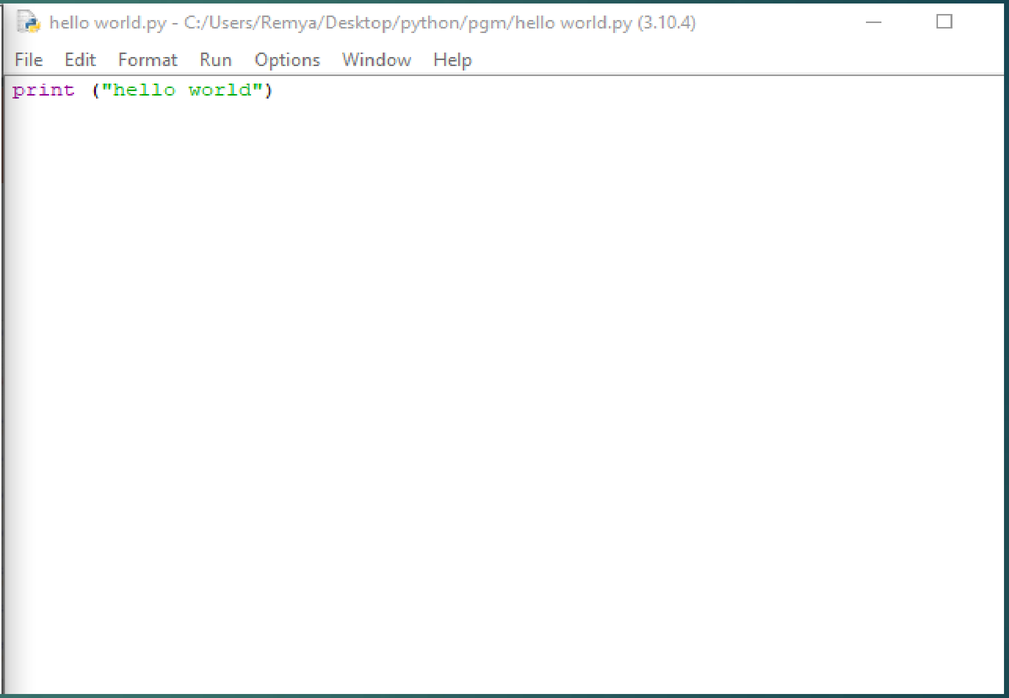
C:\Windows\system32\cmd.exe - python

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\dell>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

PYTHON IDLE



```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=========== RESTART: C:/Users/Remya/Desktop/python/pgm/hello world.py ==========
hello worls
>>>
=========== RESTART: C:/Users/Remya/Desktop/python/pgm/hello world.py ==========
hello world
>>>
```


hello world.py - C:/Users/Remya/Desktop/python/pgm/hello world.py (3.10.4)
File Edit Format Run Options Window Help

```
print ("hello world")
```

IDLE(Integrated Development and Learning Environment)

1.Open IDLE by clicking the application icon

2.Open File menu and click New File and type your first script (Eg: print("welcome to Python")

3.Save your file with .py extension (Eg:test.py)

4.From the **Run menu** click the **Run Module(** or press F5-short cut).This will run the script

5.From the File menu choose Exit to quit from IDLE

- IDLE is Python’s Integrated Development and Learning Environment.
- IDLE has the following features:
  - cross-platform: works mostly the same on Windows, Unix, and macOS
  - Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
  - multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features

IDLE has two main window types, the Shell window and the Editor window.

It is possible to have multiple editor windows simultaneously.

Jupyter Notebook

- Jupyter Notebook is a web application that allows you to create and share documents that contain:
  live code (e.g. Python code)
  visualizations
  explanatory text (written in markdown syntax)

  Jupyter Notebook is great for the following use cases:
  learn and try out Python
  data processing / transformation
  numeric simulation
  statistical modeling
  machine learning
  Jupyter Notebook is perfect for using Python for scientific computing and data analysis with libraries like numpy, pandas, and matplotlib.

Setting Up Jupyter Notebook

- **Setting Up Jupyter Notebook**
  The first step to get started is to visit the project's website at http://www.jupyter.org:
-

  Here you'll find two options:
  Try it in your browser
  Install the Notebook

DETECTING AND CORRECTING SYNTAX ERRORS

- Syntax errors are defined as violations of rules and regulations to form a layout of a certain logic.
- Syntax of tools are the structures and the building blocks to program any software.
- Errors in the syntax are the most common type of occurring errors in any programming language, especially if one is not familiar with it.

▶ Reserved keywords, built-in functions, spaces, punctuations, and other semantics required, to use python's tools needs to be strictly written as they are advised to. If any violations in the syntax and your program will not compile.

When you write your code, the interpreter compiles and converts the code into a format that can be understood by your machine. The code cannot be construed and parsed if there are any **invalid syntax errors.**

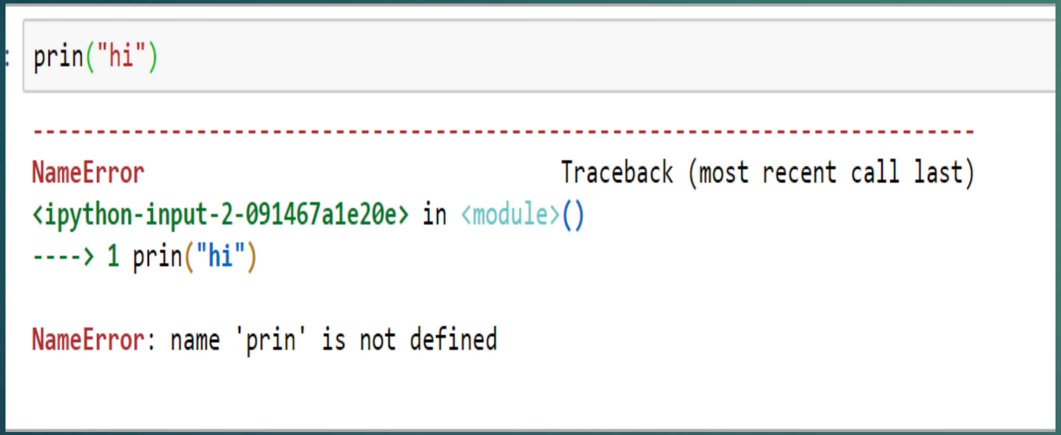
- Syntax errors are detected while the program is being compiled, once any error is found, it will prevent the code from executing.
- Usually, the errors are self-explanatory and doesnt needs any special attention to fix them.While some errors are not as corporative.
- The good thing about syntax errors is that compiler points out to where the problem might be.

Let’s look at some most common causes of syntax errors.

1. Misspelled reserved keywords
2. Missing required spaces
3. Missing quotes
4. Misuse of block statements (if-else, loops)
5. Missing assignment operator (=)
6. Invalid variables declaration
7. Invalid function calling or defining

MISSPELLED RESERVED WORDS

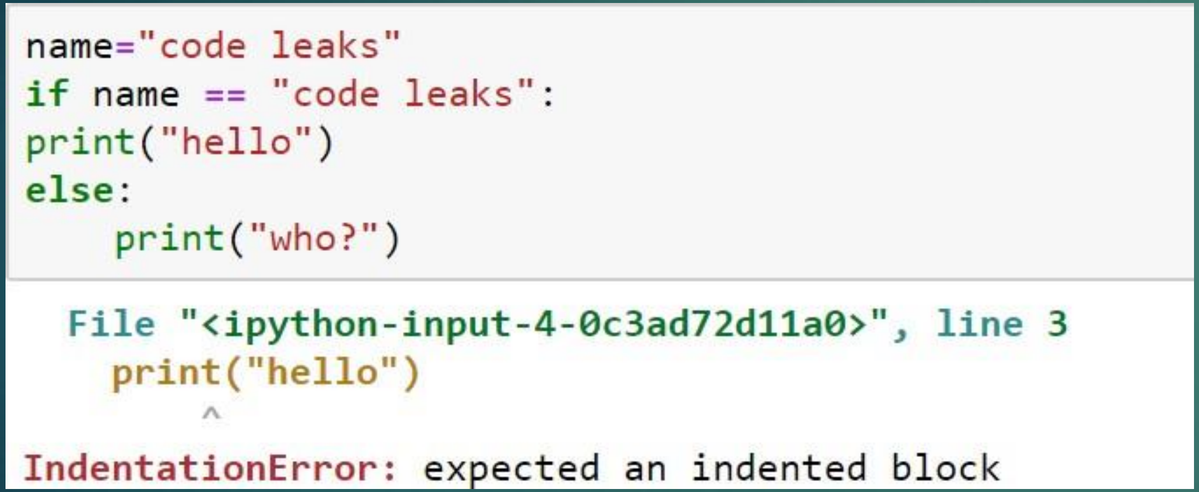
The compiler threw an error message as “prin not defined”. It’s not defined as a user-defined or built-in keyword, therefore it confuses the compiler to where this word lies.



```
prin("hi")

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-091467a1e20e> in <module>()
----> 1 prin("hi")

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

- **MISSING REQUIRED SPACES**

- Unlike other programming languages, python has the requirement for an indented block. That is why many programmers have trouble wrapping this concept in the early stages.

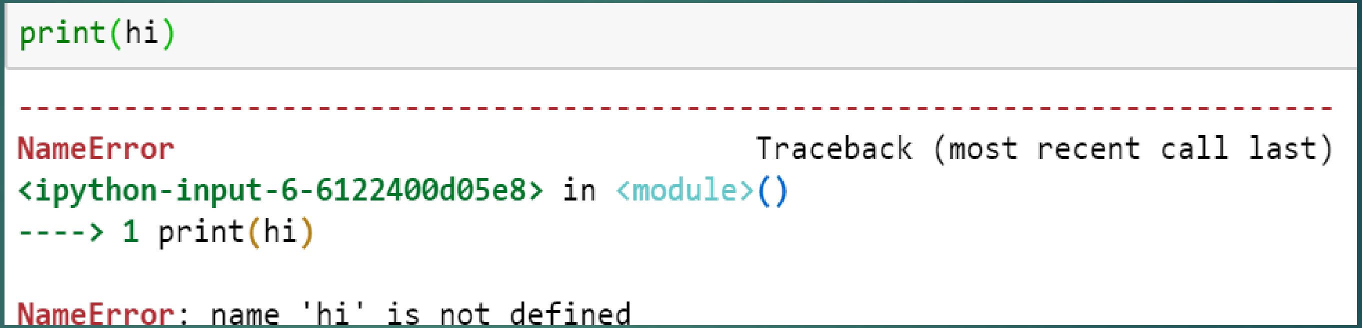


```
name="code leaks"
if name == "code leaks":
print("hello")
else:
    print("who?")
```

```
  File "<ipython-input-4-0c3ad72d11a0>", line 3
    print("hello")
        ^
IndentationError: expected an indented block
```

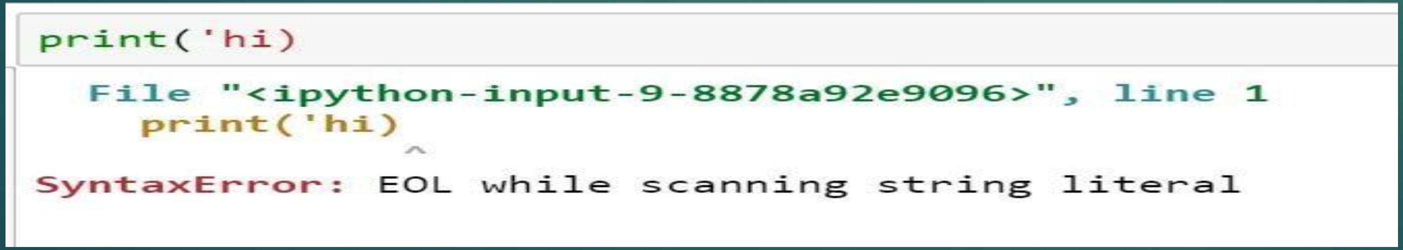
MISSING QUOTES

When missing quotes in a string, the compiler confuses the purpose of the string and doesn't identify it. Note how the error is "name hi not defined", even though it's supposed to be a string, not a variable. The compiler mistook it as a variable not defined and nowhere guessed the possibility of a string



```
print(hi)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-6122400d05e8> in <module>()
----> 1 print(hi)

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

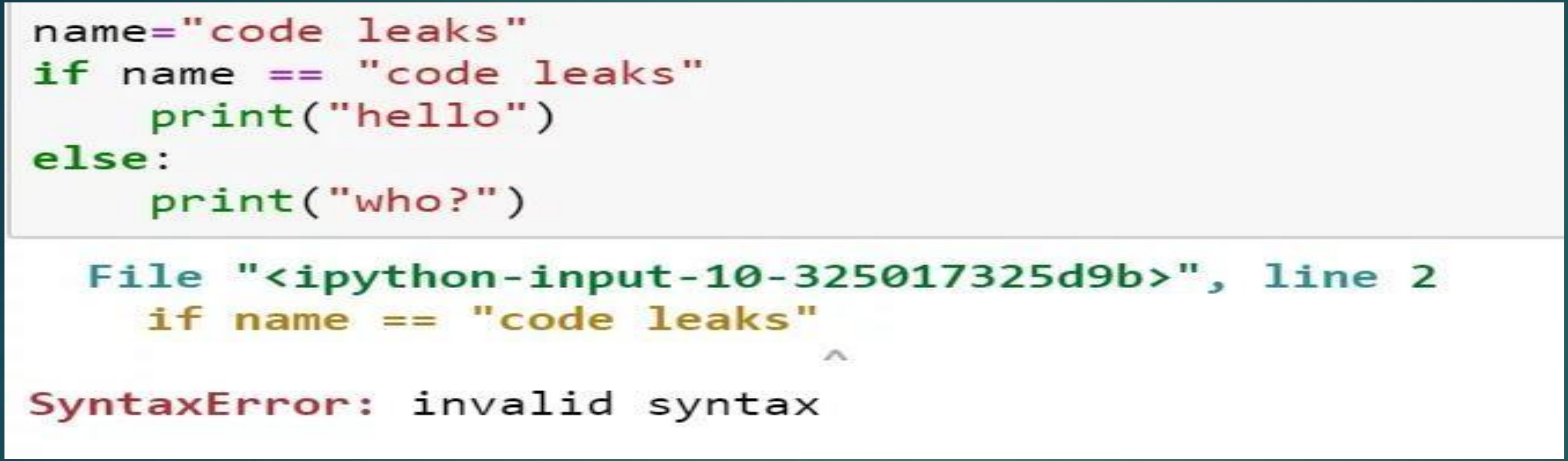
Notice how the error changed to the literal error when a quote is added. The compiler recognized it as a string and END OF LINE error is thrown.



```
print('hi)

  File "<ipython-input-9-8878a92e9096>", line 1
    print('hi)
             ^
SyntaxError: EOL while scanning string literal
```

MISUSE OF BLOCK STATEMENTS (IF-ELSE, LOOPS)

This is similar to missing spaces in addition to the missing semicolon (:). Python has another rule to use a (:) while ending block statements like loops, if-else.

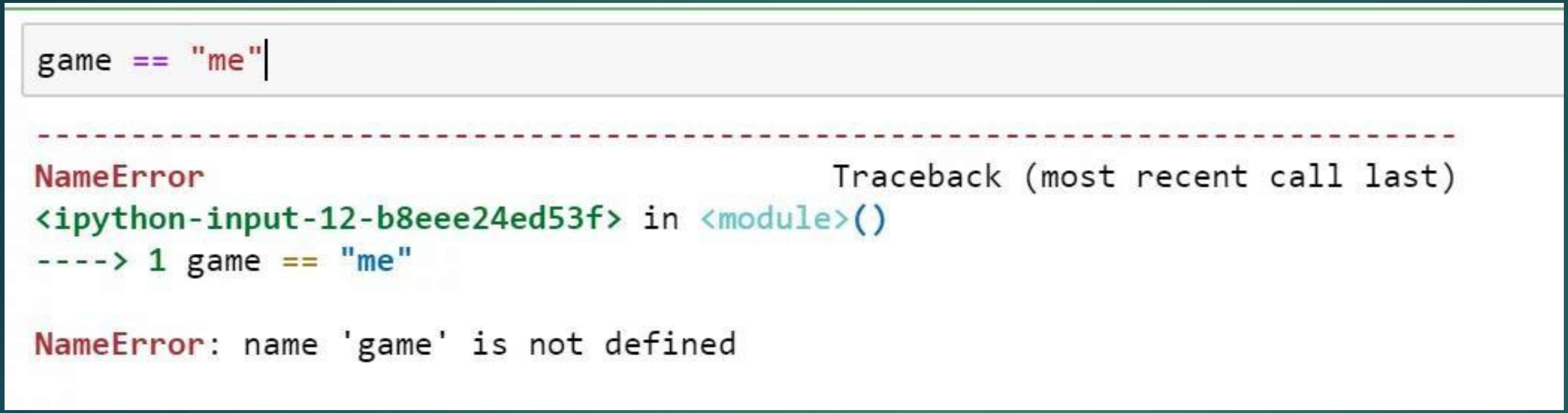


```
name="code leaks"
if name == "code leaks"
     print("hello")
else:
     print("who?")
```

```
  File "<ipython-input-10-325017325d9b>", line 2
    if name == "code leaks"
                           ^
SyntaxError: invalid syntax
```

The error is “invalid syntax”. Not very descriptive, that is why an **if-else colon (:)** is required to fix this bug. It’s the same with while blocks.

MISSING ASSIGNMENT OPERATOR (=)

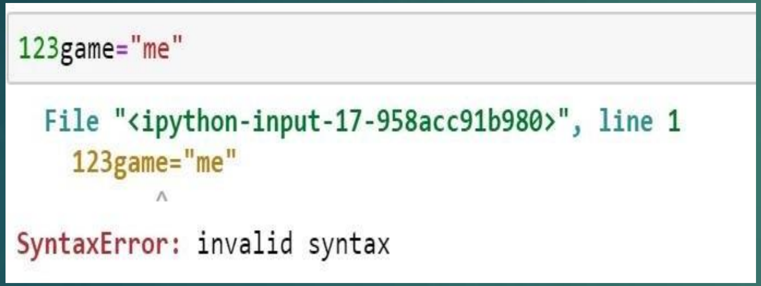


```
game == "me"

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-b8eee24ed53f> in <module>()
----> 1 game == "me"

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

Notice that the compiler is not throwing an error to indicate that the assignmentoperator is being misused. But it’s trying to compare the variable game to the string “me”, finding undefined variable.

INVALID VARIABLES DECLARATION

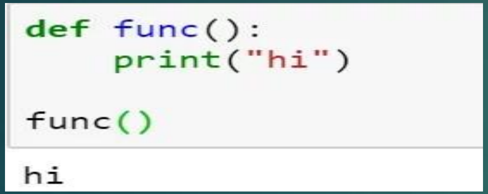
There are many ways to violate the variable naming convention. You cannot use special characters to expect underscore (_), or use a number at the beginning variable and many others.



```
123game="me"

  File "<ipython-input-17-958acc91b980>", line 1
    123game="me"
          ^
SyntaxError: invalid syntax
```

INVALID FUNCTION CALLING OR DEFINING

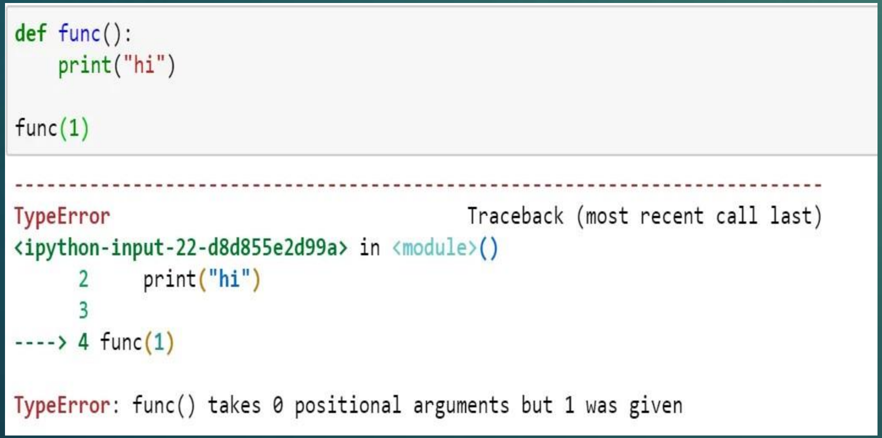
Like any other block statement, function declaration has also a syntax. Proper spaces and use of the colon (:) are necessary. Messing up with the syntax will prevent from execution. The following example shows an executable function without errors.



```
def func():
    print("hi")

func()
```

```
hi
```

Function calling has to have the required cautions to prevent bugs. The following error is caused by the argument provided which is not defined in the declaration.



```
def func():
    print("hi")

func(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-d8d855e2d99a> in <module>()
      2     print("hi")
      3
----> 4 func(1)

TypeError: func() takes 0 positional arguments but 1 was given
```

Output and Input functions- print() and input()

- We use the print() function to output data to the standard output device (screen)
  An example of its use is given below.
  >>>print('This is a sample output')
  This is a sample output
- Another example is given below:
  >>>x = 5
- >>>print('The value of x is', x)

  The value of x is 5

Print()

- print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

  Here, objects is the value(s) to be printed.
- The sep separator is used between the values. It defaults into a space character.
- After all values are printed, end is printed. It defaults into a new line.
- The file is the object where the values are printed and its default value is sys.stdout (screen).
-

Print()

```
>>> print(1,2,3)
1 2 3
>>> print(1,2,3,sep="**")
1**2**3
>>> print(1,2,3,sep="**",end="$")
1**2**3$>>>
```

input()

The syntax for input() is:

input([prompt])

where prompt is the string we wish to display on the screen. It is optional.

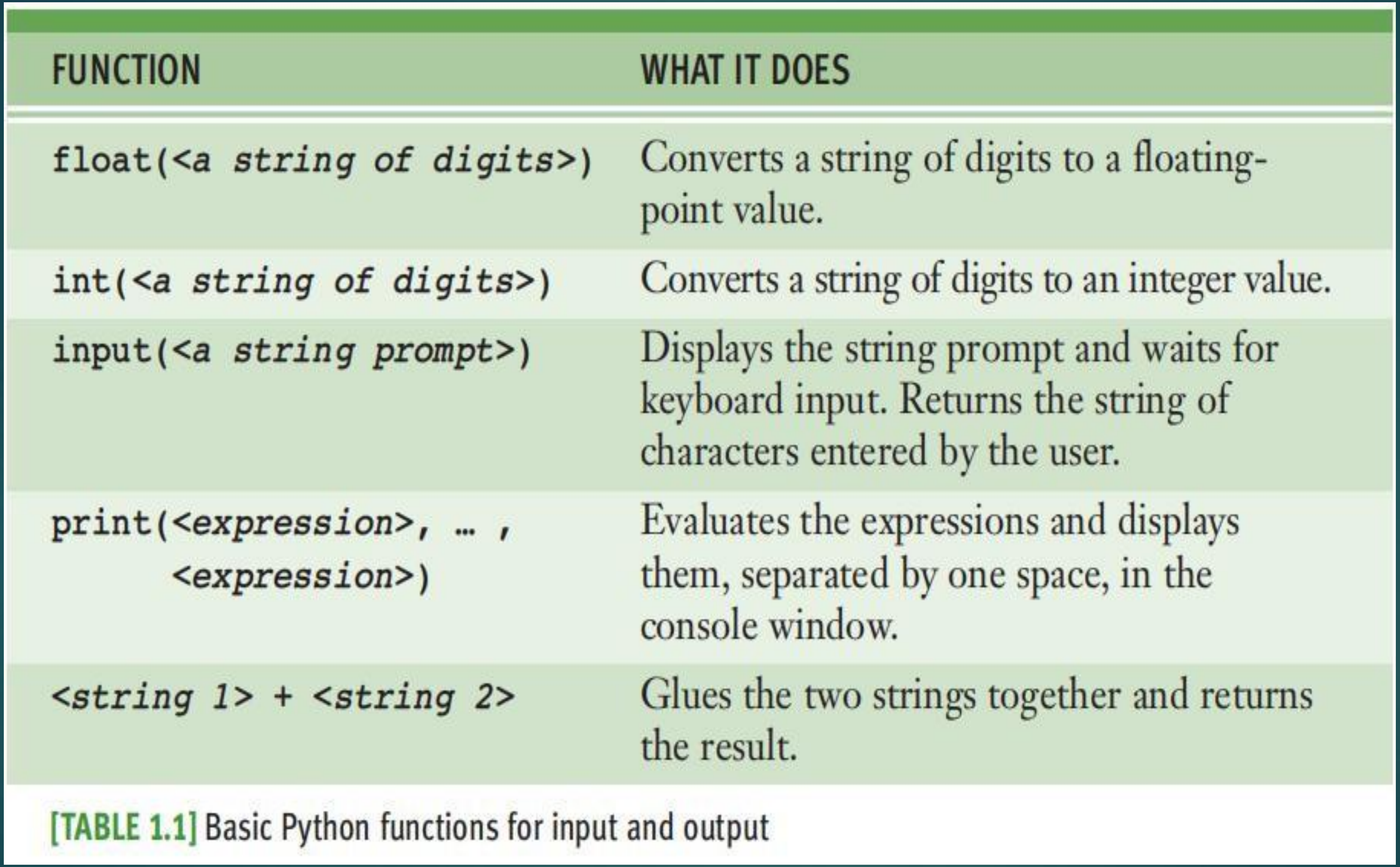
```
>>> num = input('Enter a number: ')
>> x=input("enter a number")
enter a number3
>>> print(x)
3
>>> print(type(x))
<class 'str’>
```

Input()

- Here, we can see that the entered value 3 is a string, not a number. To convert this into a number we can use int() or float() functions.

```
>>> int(‘3’)
3
>>> float(‘3')
10.0
```

```
>>> first = int(input(“Enter the first
number: “))
Enter the first number: 23
>>> second = int(input(“Enter the second
number: “))
Enter the second number: 44
>>> print(“The sum is”, first +
second)
The sum is 67
>>>
```



FUNCTION	WHAT IT DOES
`float(<a string of digits>)`	Converts a string of digits to a floating-point value.
`int(<a string of digits>)`	Converts a string of digits to an integer value.
`input(<a string prompt>)`	Displays the string prompt and waits for keyboard input. Returns the string of characters entered by the user.
`print(<expression>, … , <expression>)`	Evaluates the expressions and displays them, separated by one space, in the console window.
`<string 1> + <string 2>`	Glues the two strings together and returns the result.

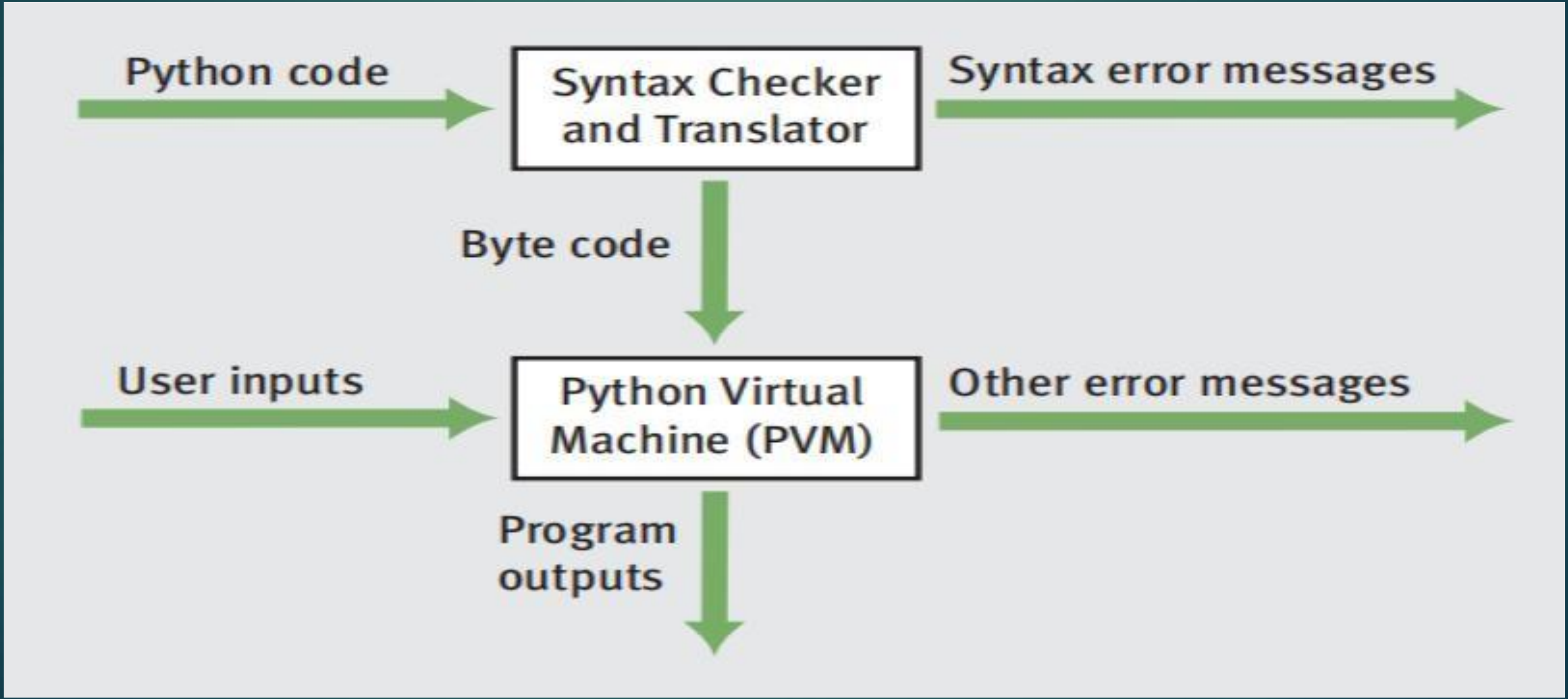
[TABLE 1.1] Basic Python functions for input and output

Editing, Saving, and Running a Script

To compose and execute programs without opening IDLE, you perform the following steps:

1. Select the option New Window from the File menu of the shell window.
2. In the new window, enter Python expressions or statements on separate lines, in the order in which you want Python to execute them.
3. At any point, you may save the file by selecting File/Save. If you do this, you should use a .py extension.
4. To run this file of code as a Python script, select Run Module from the Run menu or press the F5 key (Windows).

Behind the Scenes: How Python Works



Steps in interpreting a Python program

1)The interpreter reads a Python expression or statement, also called the source code, and verifies that it is well formed.

2)If a Python expression is well formed, the interpreter then translates it to an equivalent form in a low-level language called byte code.

3)This byte code is next sent to another software component, called the Python virtual machine (PVM), where it is executed. If another error occurs during this step, execution also halts with an error message.

The software development process - Case Study.

Waterfall Model - Different Phases

1.Customer request—In this phase, the programmer receives a broad statement a problem to be solved. This is the user requirement specification phase

2.Analysis—The programmers determine what the program will do. This is sometimes viewed as a process of clarifying the specifications for the problem.

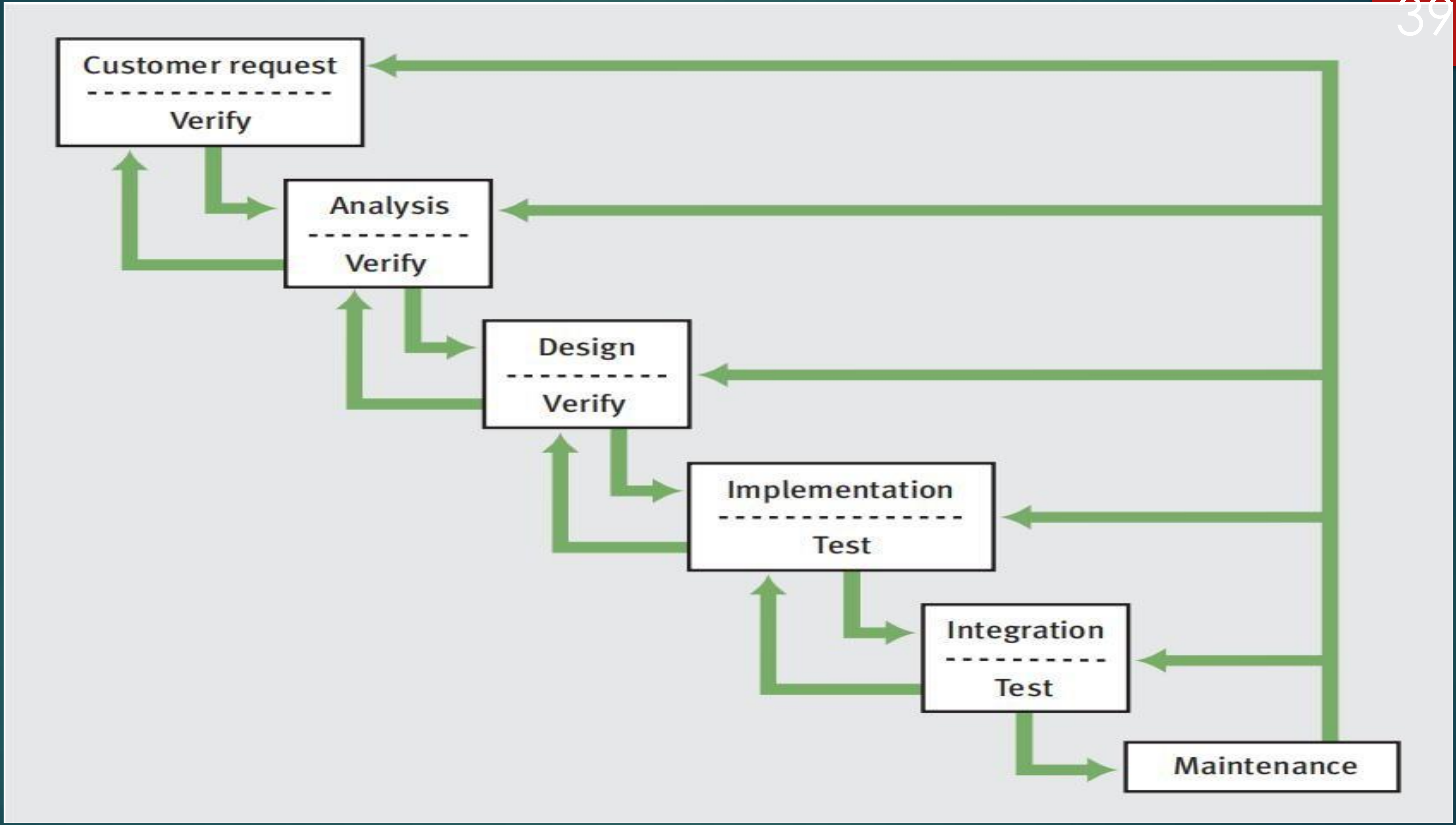
3. Design—The programmers determine how the program will do its task.

4) Implementation—The programmers write the program.

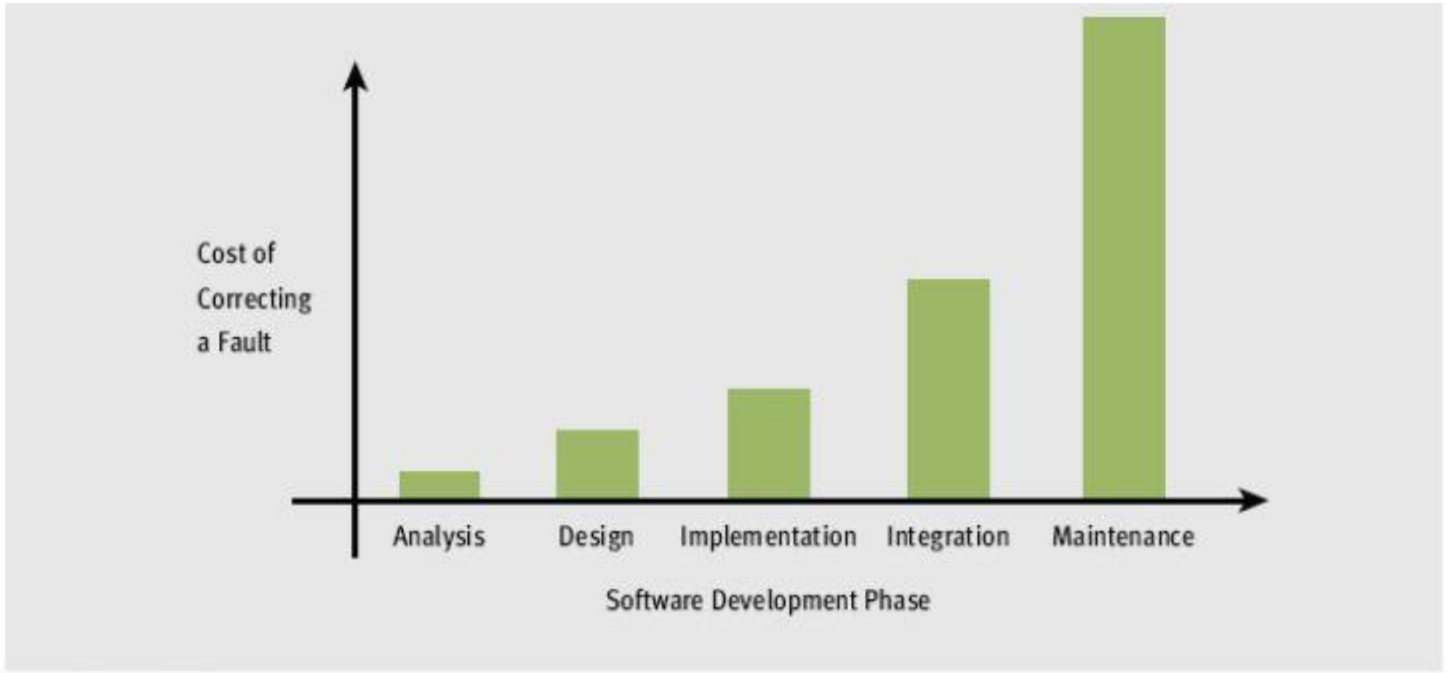
This step is also called the coding phase.

5)Integration—Large programs have many parts. In the integration phase, these parts are brought together into a smoothly functioning whole, usually not an easy task.

6)Maintenance—Programs usually have a long life; a lifespan of 5 to15 years is common for software. During this time, requirements change,errors are detected, and minor or major modifications are made

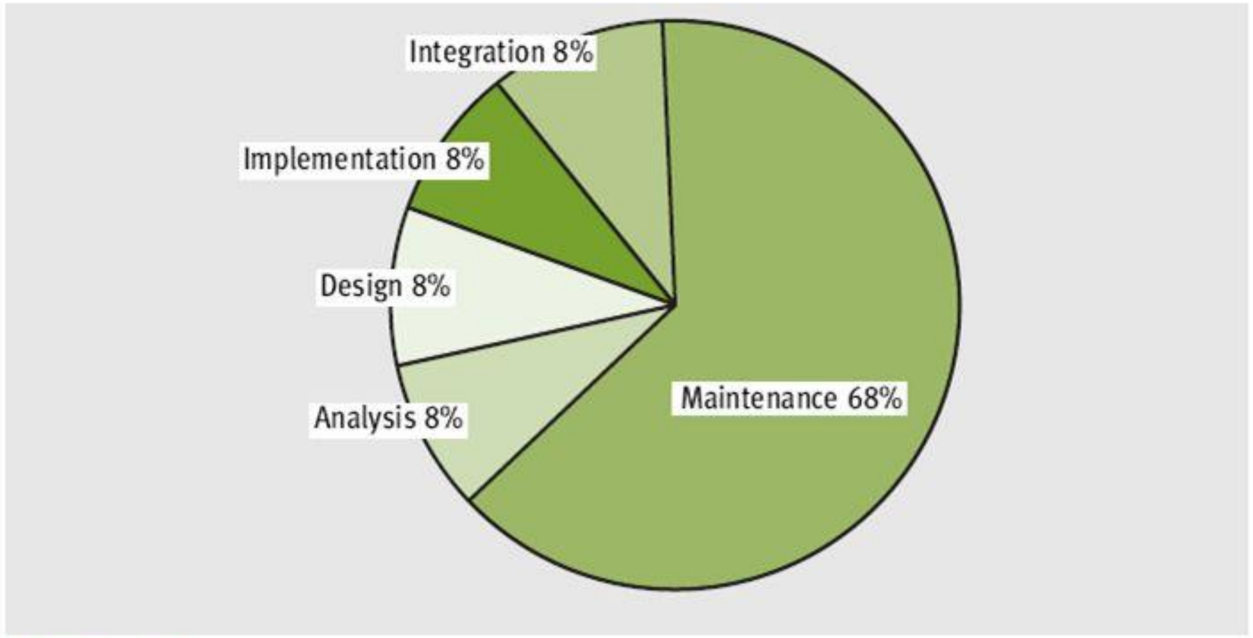

Customer request
Verify
Analysis
Verify
Design
Verify
Implementation
Test
Integration
Test
Maintenance

The Software Development Process (continued)



[FIGURE 2.2] Relative costs of repairing mistakes that are found in different phases

The Software Development Process (continued)



[FIGURE 2.3] Percentage of total cost incurred in each phase of the development process

CASE STUDY : INCOME TAX CALCULATOR

Request:

The customer requests a program that computes a person's income tax.

Analysis:

Analysis often requires the programmer to learn some things about the problem domain, in this case, the relevant tax law.

- ✓ All taxpayers are charged a flat tax rate of 20%.
- ✓ All taxpayers are allowed a $10,000 standard deduction.
- ✓ For each dependent, a taxpayer is allowed an additional $3,000 deduction.
- ✓ Gross income must be entered to the nearest penny.
- ✓ The income tax is expressed as a decimal number.

Another part of analysis determines what information the user will have to provide. user inputs - gross income and number of dependents.

- Design:
- we describe how the program is going to do it.
- This usually involves writing an algorithm - pseudocode

```
1) Input the gross income and number of dependents
2) Compute the taxable income using the formula
3) Taxable income = gross income - 10000 - (3000 * number of dependents)
4) Compute the income tax using the formula
5) Tax = taxable income * 0.20
6) Print the tax
```

Implementation (Coding):

Given the preceding pseudocode, an experienced programmer would now find it easy to write the corresponding Python program.

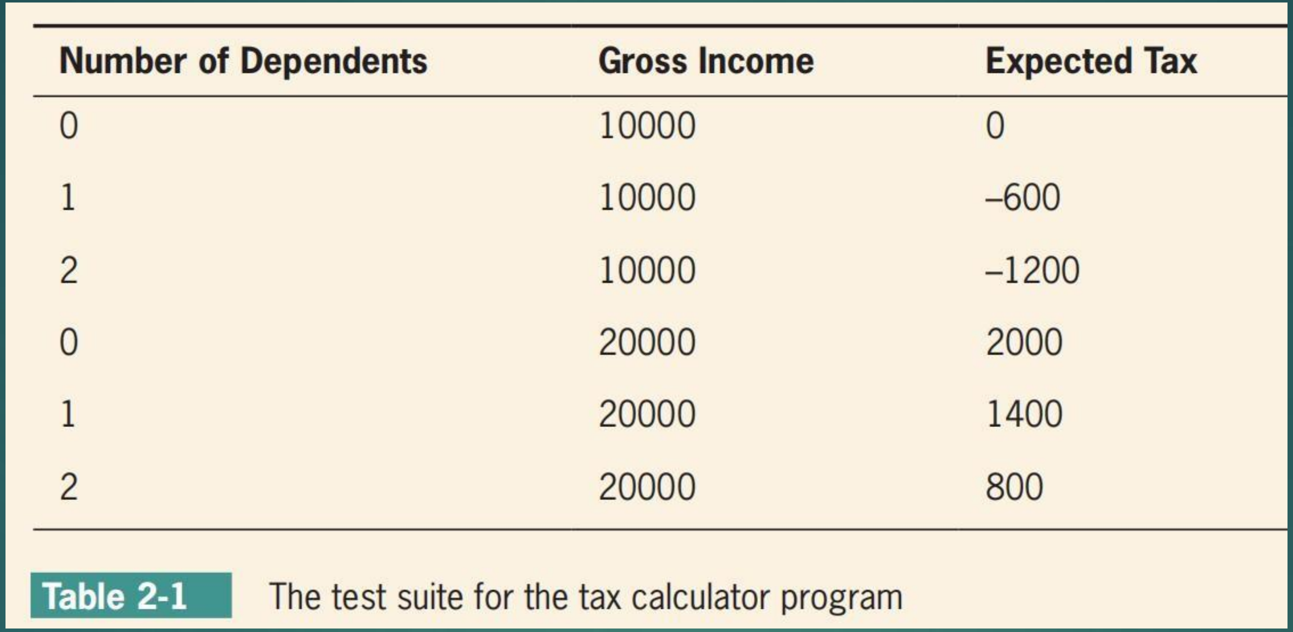
Testing:

Only thorough testing can build confidence that a program is working correctly. Testing is a deliberate process that requires some planning and discipline on the programmer's part.

A correct program produces the expected output for any legitimate input.

Testing all of the possible combinations of inputs would be impractical. The challenge is to find a smaller set of inputs, called a test suite, from which we can conclude that the program will likely be correct for all inputs.

```
# Initialize the constants
TAX_RATE = 0.20
STANDARD_DEDUCTION = 10000.0
DEPENDENT_DEDUCTION = 3000.0
# Request the inputs
grossIncome = float(input("Enter the gross income: "))
numDependents = int(input("Enter the number of dependents: "))
# Compute the income tax
taxableIncome = grossIncome - STANDARD_DEDUCTION (
 DEPENDENT_DEDUCTION * numDependents)
incomeTax = taxableIncome * TAX_RATE
# Display the income tax
print("The income tax is $" + str(incomeTax))
```

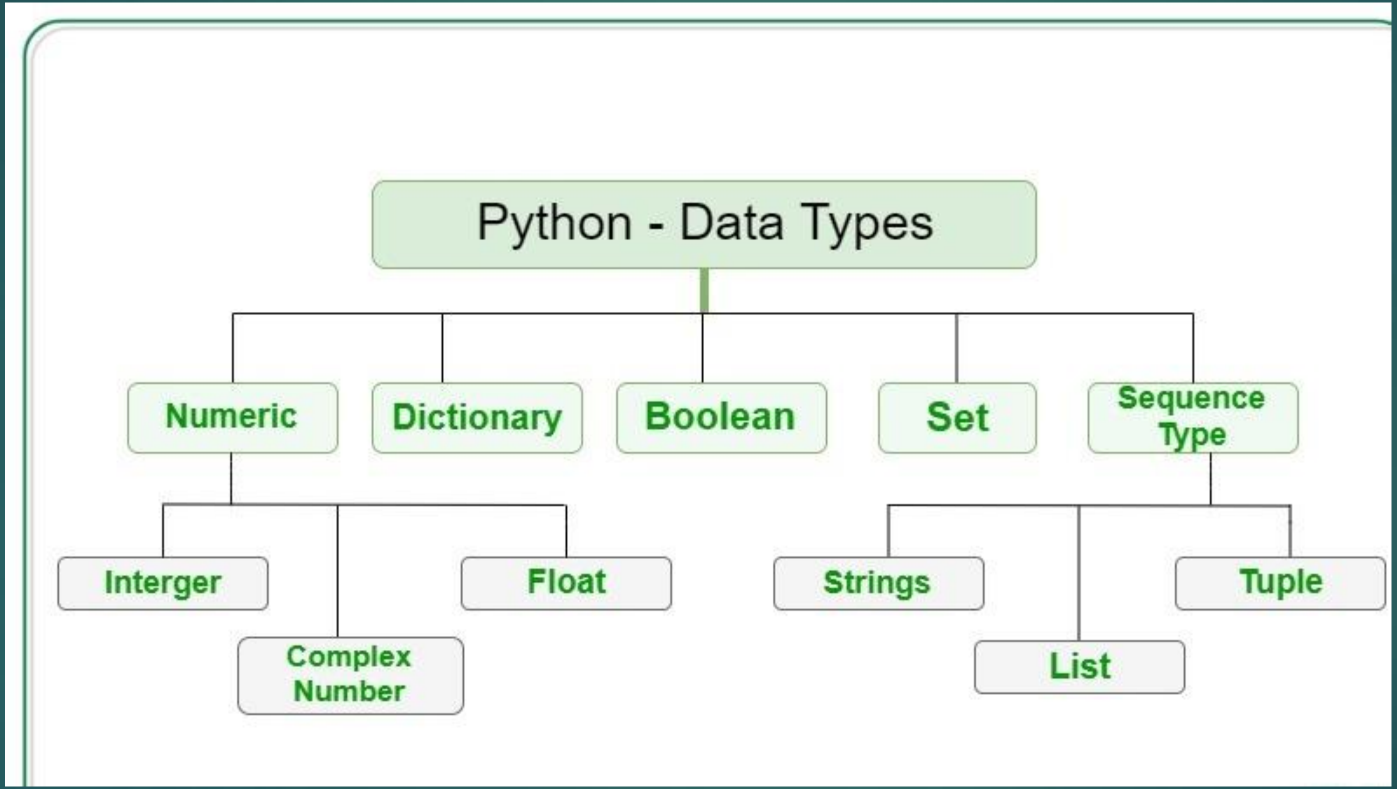


Number of Dependents	Gross Income	Expected Tax
0	10000	0
1	10000	–600
2	10000	–1200
0	20000	2000
1	20000	1400
2	20000	800

Table 2-1 The test suite for the tax calculator program

If there is a logic error in the code, it will almost certainly be caught using these data.

DATATYPES IN PYTHON



NUMERIC DATATYPES

1. **Int -** Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to **int**
2. **Float -** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.
3. **complex -** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Strings

- In Python, Strings are arrays of bytes representing Unicode characters.
- A string is a collection of one or more characters put in a single quote, double-quote or triple quote.
- In python there is no character data type, a character is a string of length one. It is represented by str class.

String Literals

- ✓ In Python, a string literal is a sequence of characters enclosed in single or double quotation marks.

```
>>> 'Hello there!'
'Hello there!'
>>> "Hello there!"
'Hello there!'
```

- ✓ To output a paragraph of text that contains several lines

```
>>> print("""This very long sentence
extends  all the way to the next line.""")
This very long sentence extends
all the way to the next line.
```

3. List:- It is the type that defines sequence of data. It allows one to add, delete or process elements in very simple ways. It is similar to arrays in C.

Syntax:- List_var=[Val1,Val2,Val3,…]

Each element in the list is separated by comma and enclosed by a pair of square brackets. Elements in eth list can be of different type.

Eg:- list1=[“this is a string”,12], here both string and numeric type data is placed as the contents of list

4. Tuples:- It is similar to list, holding a sequence of data. It consists of values separated by commas.

Tuples are enclosed in parenthesis.

In list elements are enclosed in [] and the elements and size can be changed.

But in tuple elements are enclosed in () and the elements cannot be updated.

Eg:- tuple=('abcd',786,2.23), here tuple is created with different types of values

5. Dictionary:-

It is used to store key value pairs. It enables us to quickly retrieve, add, remove, modify , values

using key.

The dictionary key can be any python type usually strings.

They can be created using pair of curly braces {}.

Each item in the dictionary consist of a key, followed by a colon and which is followed by a avalue. Each key of the dictionary must be unique.

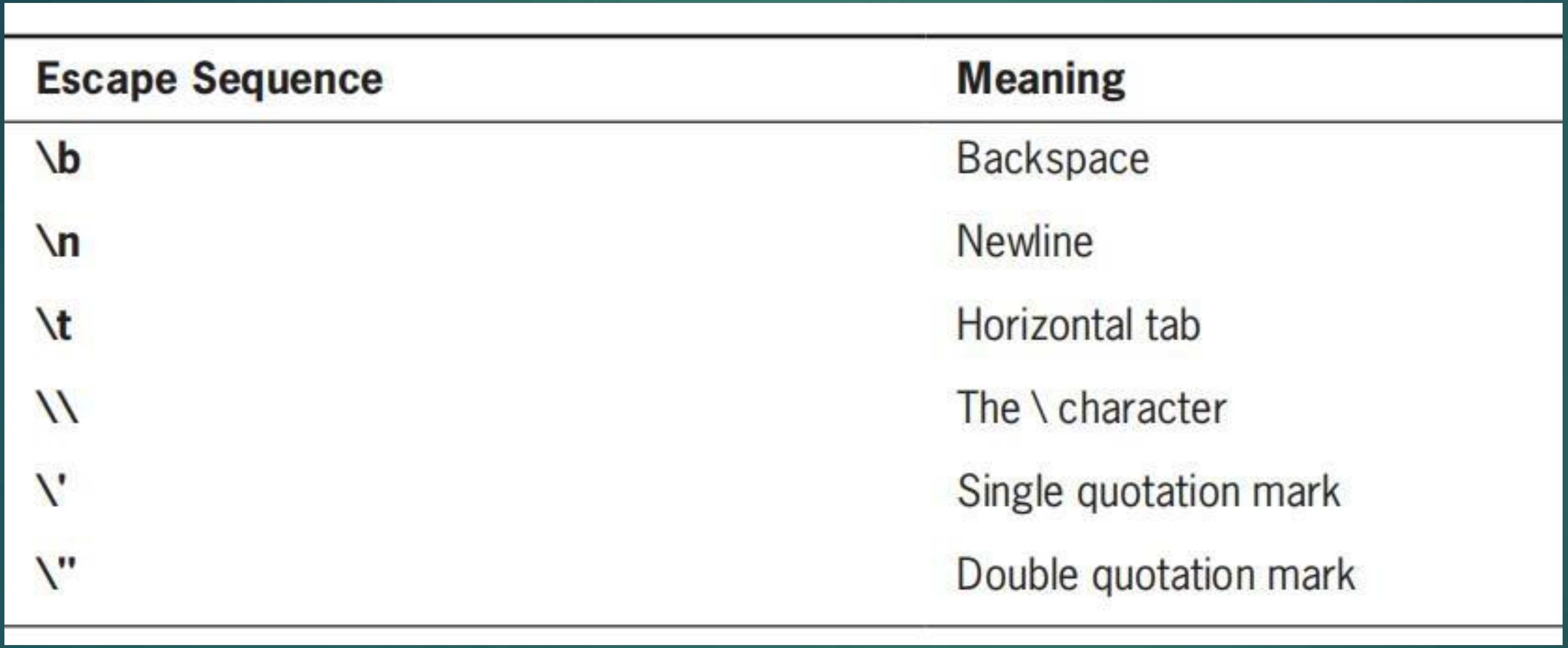
Syntax:- Dict_var={key1:val1,key2:val2,…}

Eg:

Friends = { ‘ron’:’111-222-333’,’joy’:’666-33-111’}

Escape Sequences

Escape sequences are the way Python expresses special characters, such as the tab, the newline, and the backspace (delete key), as literals.



Escape Sequence	Meaning
\b	Backspace
\n	Newline
\t	Horizontal tab
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark

Identifiers

An identifier is a name given to entities like variables,functions,class etc. It helps to differentiate one entity from another.

Rules for writing identifiers

1.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 _. Names like myClass, var_1 and print_this_to_screen, all are valid example.

2.An identifier cannot start with a digit. 1variable is invalid, but variable1 is a valid name.

3.Keywords cannot be used as identifiers.

Eg: global = 1 is invalid

4.We cannot use special symbols like !, @, #, $, % etc. in our identifier.a@=0 is invalid

5.An identifier can be of any length.

Things to Remember

-
  Python is a case-sensitive language. This means, variable X and x are not the same.
- Always give the identifiers a name that makes sense.
- Multiple words can be separated using an underscore, like this_is_a_long_variable.

variables

- A variable can be used to store a certain value or object. In Python, all numbers (and everything else, including functions) are objects. A variable is created through assignment. Their type is assigned dynamically.
  Eg:
  x=‘name'
  y=23
  z=24.5
  l=399379379379387983793773973977000102

assignment statement

<variable name> = <expression>

The Python interpreter first evaluates the expression on the right side of the assignment symbol and then binds the variable name on the left side to this value. When this happens to the variable name for the first time, it is called defining or initializing the variable.

type(object)

type(object) to know the type of the variable object

Eg: type(x)   - <type 'str'>

type(y)   - <type 'int'>

type(z)   -<type 'float'>

type(l) - <type 'long'>

Keywords

- Keywords are the reserved words in Python.

  We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

  In Python, keywords are case sensitive.

  There are 33 keywords in Python 3.7. This number can vary slightly over the course of time.

PYTHON KEYWORDS

```
$ python3.10
>>> help()
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

- Expressions:-

  - It is the combination of operators and operands
  - It is the combination of values, variables .
  - If one types an expression, the interpreter evaluates it and gives the result. Eg- to evaluate 1+1 to get 2,
- we do:1+1 2
- Eg:- a+b, a-b etc.

Exercises

1.Let the variable x be "cat" and the variable y be "rat". Write the values returned by the following operations:

a. x + y

b. "the " + x + " chases the " + y

c. x * 4

2. Which of the following are valid variable names?

a. length

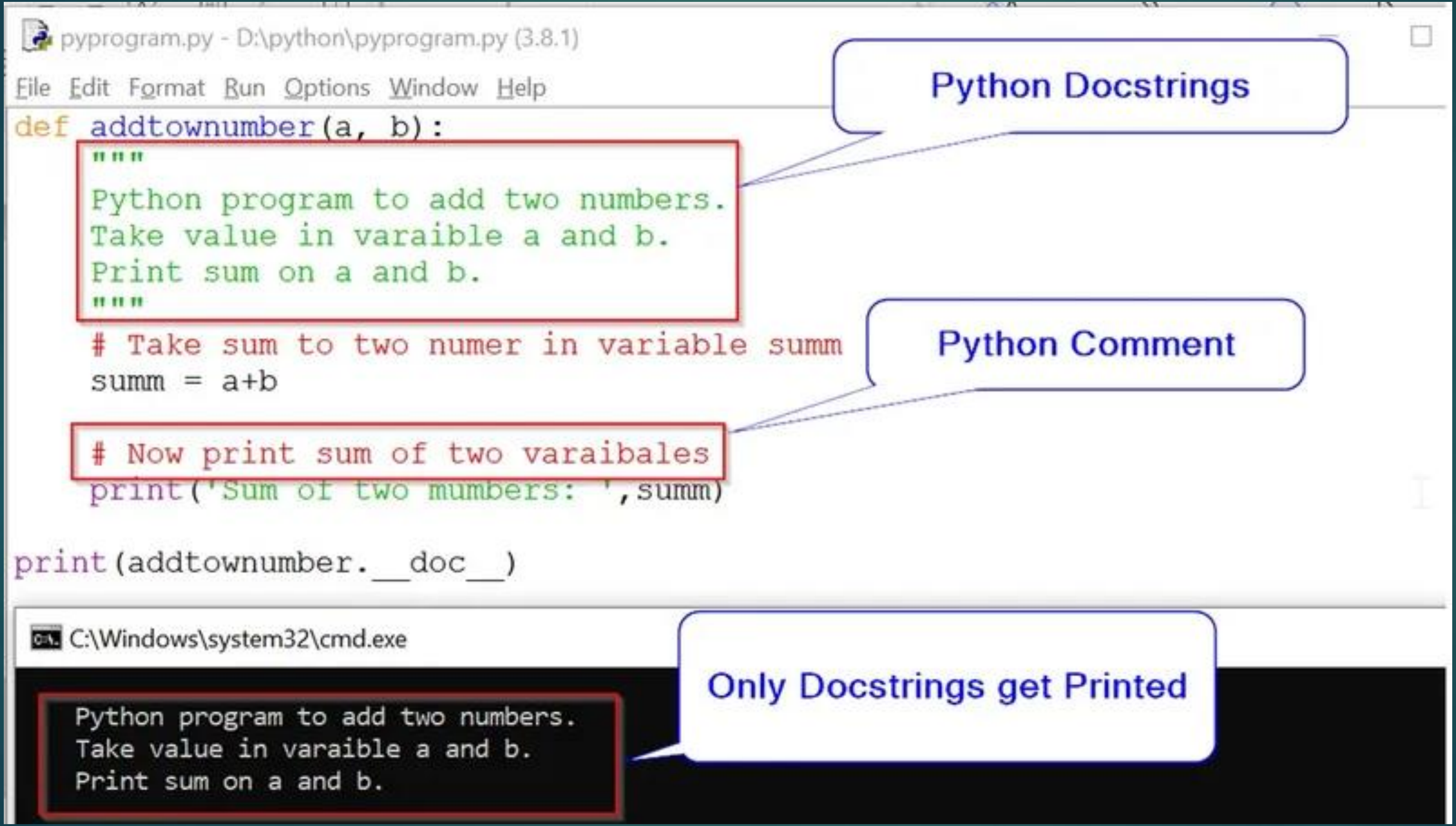
b. _width

c. firstBase

d. 2MoreToGo

e. halt!

Comments and docstrings


pyprogram.py - D:\python\pyprogram.py (3.8.1)
File Edit Format Run Options Window Help
Python Docstrings
Python Comment
C:\Windows\system32\cmd.exe
Only Docstrings get Printed

```
def addtownumber(a, b):
    """
    Python program to add two numbers.
    Take value in varaible a and b.
    Print sum on a and b.
    """
    # Take sum to two numer in variable summ
    summ = a+b

    # Now print sum of two varaibales
    print('Sum of two mumbers: ',summ)

print(addtownumber.__doc__)
```

```
Python program to add two numbers.
Take value in varaible a and b.
Print sum on a and b.
```

Numeric Data Types and Character Sets

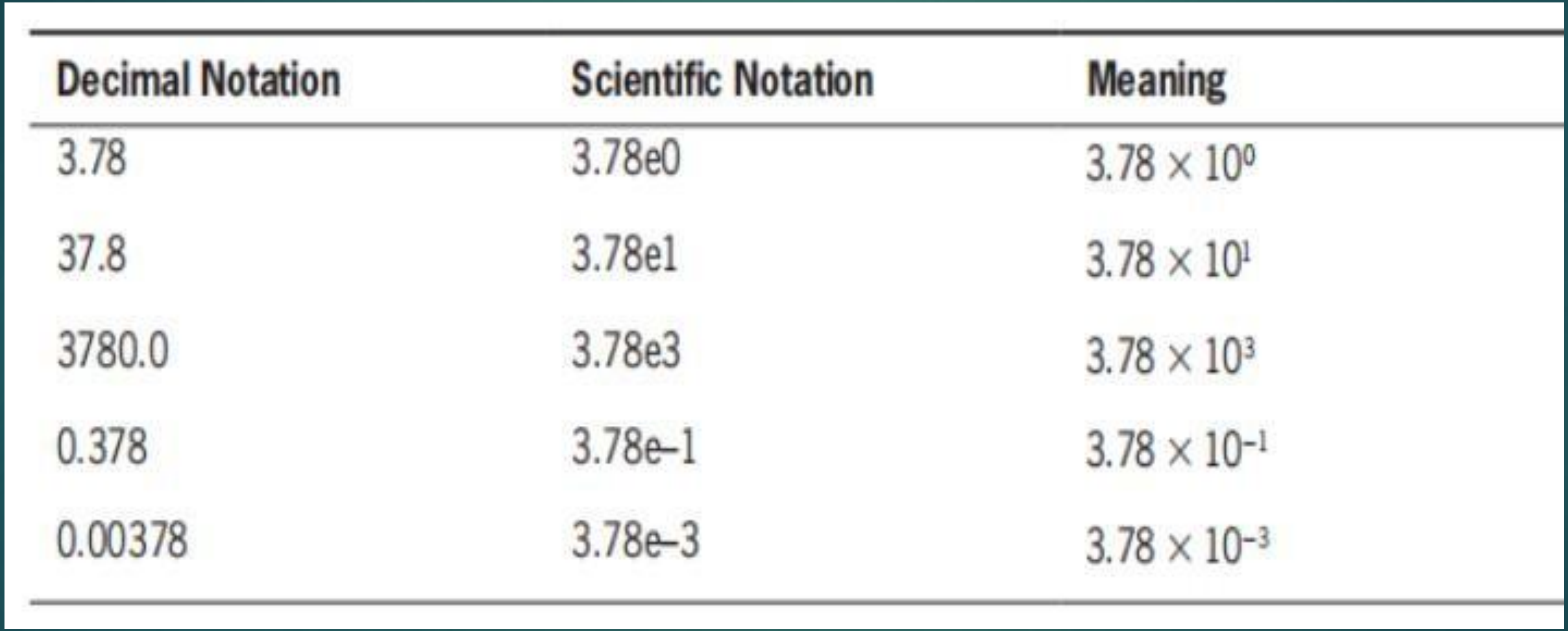
- Integers - the integers include 0, all of the positive whole numbers, and all of the negative whole numbers. range : -2^{31} to $2^{31}-1$
- Floating - Point Numbers - A real number in mathematics, such as the value of pi (3.1416...), consists of a whole number, a decimal point, and a fractional part.

  range : -10^{308} to 10^{308}
- Character Sets - ASCII set

The term ASCII stands for American Standard Code for Information Interchange.

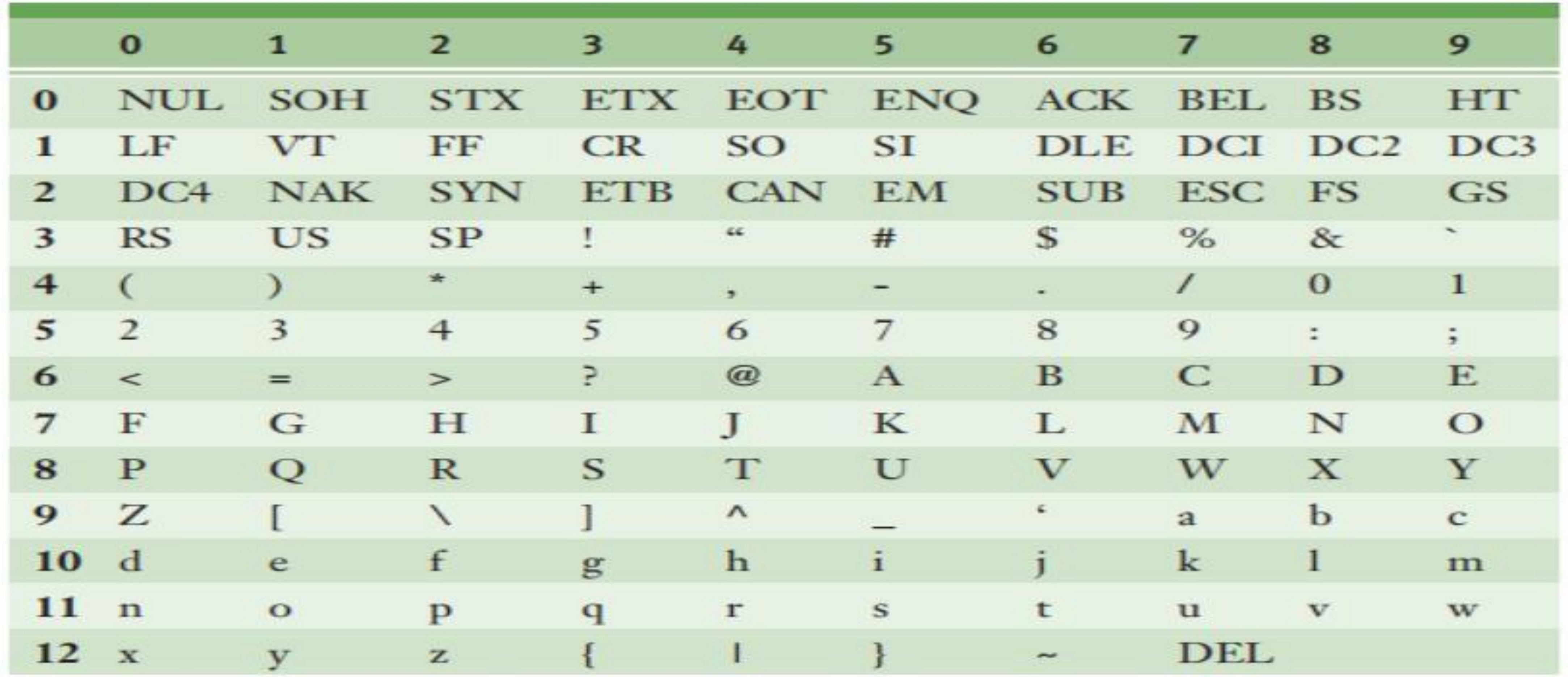
In the 1960s, the original ASCII set encoded each keyboard character and several control characters using the integers from 0 through 127.

- A floating-point number can be written using either ordinary decimal notation or scientific notation. Scientific notation is often useful for mentioning very large numbers.



Decimal Notation	Scientific Notation	Meaning
3.78	3.78e0	3.78×10^{0}
37.8	3.78e1	3.78×10^{1}
3780.0	3.78e3	3.78×10^{3}
0.378	3.78e–1	3.78×10^{-1}
0.00378	3.78e–3	3.78×10^{-3}

Numeric Data Types and Character Sets



	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DCI	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	“	#	$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

[TABLE 2.5] The original ASCII character set

Numeric Data Types and Character Sets

- The digits in the left column represent the leftmost digits of an ASCII code, and the digits in the top row are the rightmost digits. Thus, the ASCII code of the character 'R' at row 8, column 2 is 82.

- Python’s ord() and chr() functions convert characters to their numeric ASCII codes and back again, respectively.

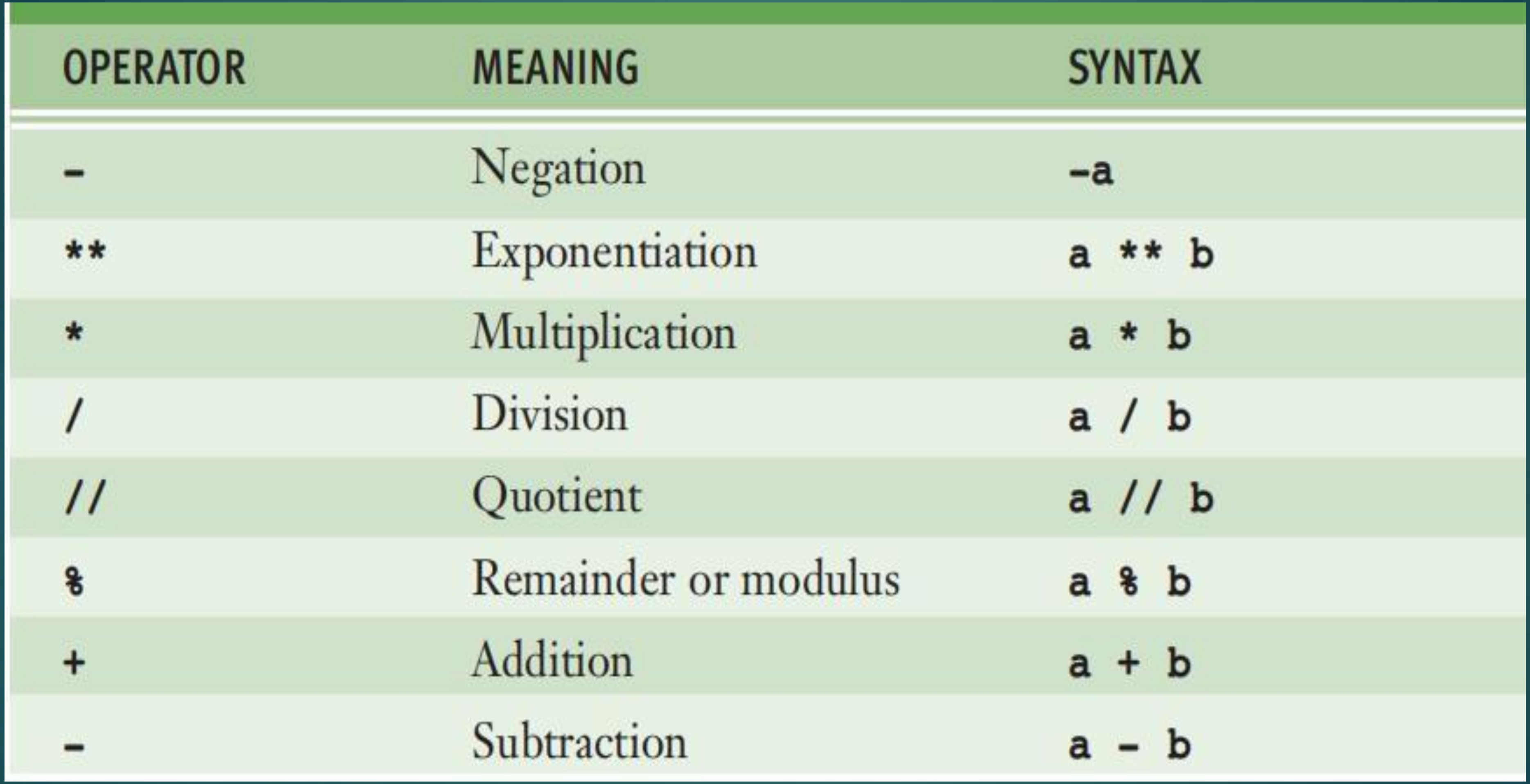
```
ord('a’)
97
>>> ord('A’)
>>> chr(66)
'B'
```

Exercises

3. Write the values of the following floating-point numbers in Python’s scientific notation:
    - a. 355.76
    - b. 0.007832
    - c. 4.3212

4. write the ASCII values of the characters '$' and '&'.

Arithmetic Expressions



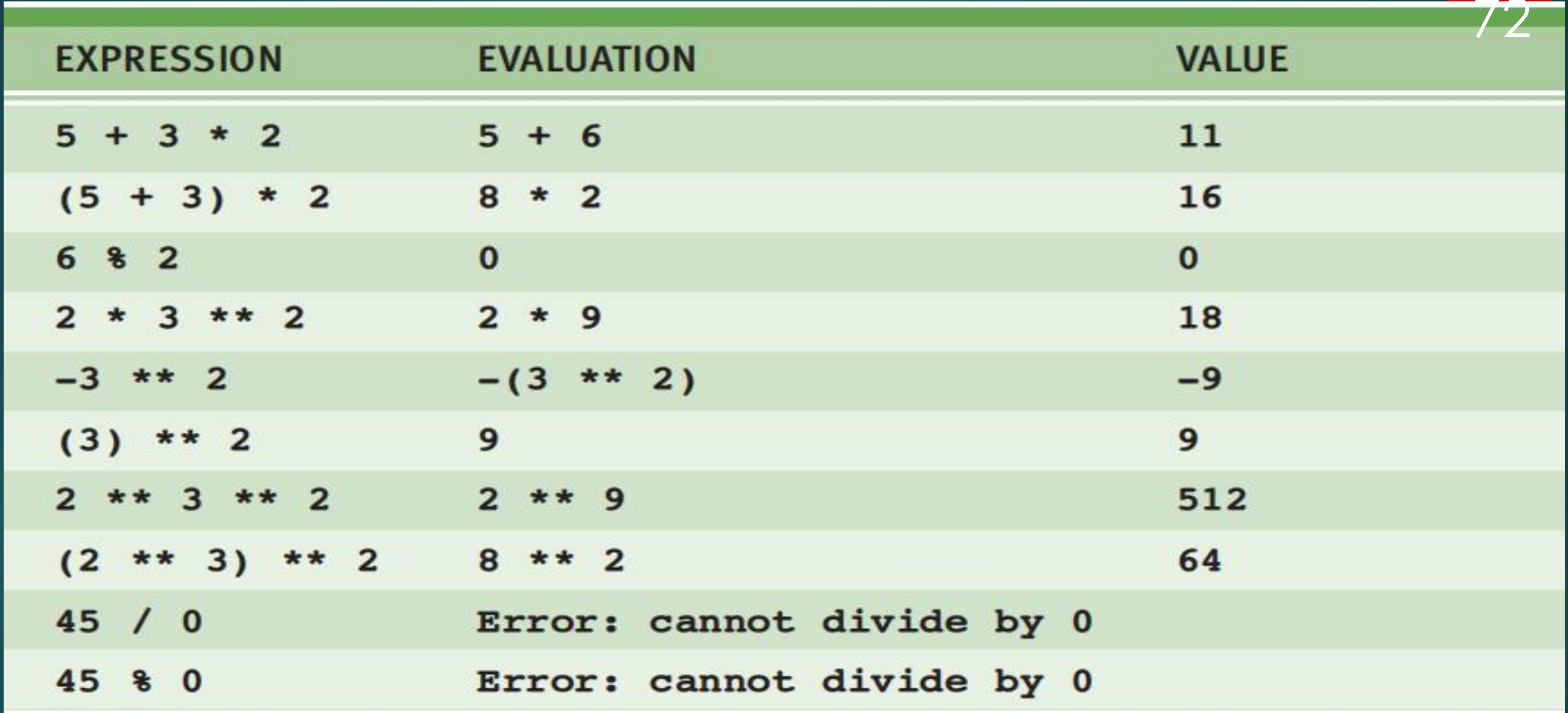
OPERATOR	MEANING	SYNTAX
`-`	Negation	`-a`
`**`	Exponentiation	`a ** b`
`*`	Multiplication	`a * b`
`/`	Division	`a / b`
`//`	Quotient	`a // b`
`%`	Remainder or modulus	`a % b`
`+`	Addition	`a + b`
`-`	Subtraction	`a - b`

precedence rules

- ✓ Exponentiation has the highest precedence and is evaluated first.
- ✓Unary negation is evaluated next, before multiplication, division, and remainder.
- ✓Multiplication, both types of division, and remainder are evaluated before addition and subtraction.
- ✓Addition and subtraction are evaluated before assignment.
- ✓With two exceptions, operations of equal precedence are left associative, so they are evaluated from left to right.

Exponentiation and assignment operations are right associative, so consecutive instances of these are evaluated from right to left.

- ✓ You can use parentheses to change the order of evaluation



EXPRESSION	EVALUATION	VALUE
`5 + 3 * 2`	`5 + 6`	`11`
`(5 + 3) * 2`	`8 * 2`	`16`
`6 % 2`	`0`	`0`
`2 * 3 ** 2`	`2 * 9`	`18`
`-3 ** 2`	`-(3 ** 2)`	`-9`
`(3) ** 2`	`9`	`9`
`2 ** 3 ** 2`	`2 ** 9`	`512`
`(2 ** 3) ** 2`	`8 ** 2`	`64`
`45 / 0`	`Error: cannot divide by 0`	
`45 % 0`	`Error: cannot divide by 0`	

[TABLE 2.7] Some arithmetic expressions and their values

- ✓Syntax is the set of rules for constructing well-formed expressions or sentences in a language.

- ✓Semantics is the set of rules that allow an agent to interpret the meaning of those expressions or sentences.

- ✓A computer generates a syntax error when an expression or sentence is not well formed.

- ✓A semantic error is detected when the action that an expression describes cannot be carried out, even though that expression is syntactically correct.

Mixed-Mode Arithmetic and Type Conversions

Performing calculations involving both integers and floating-point numbers is called mixed-mode arithmetic.

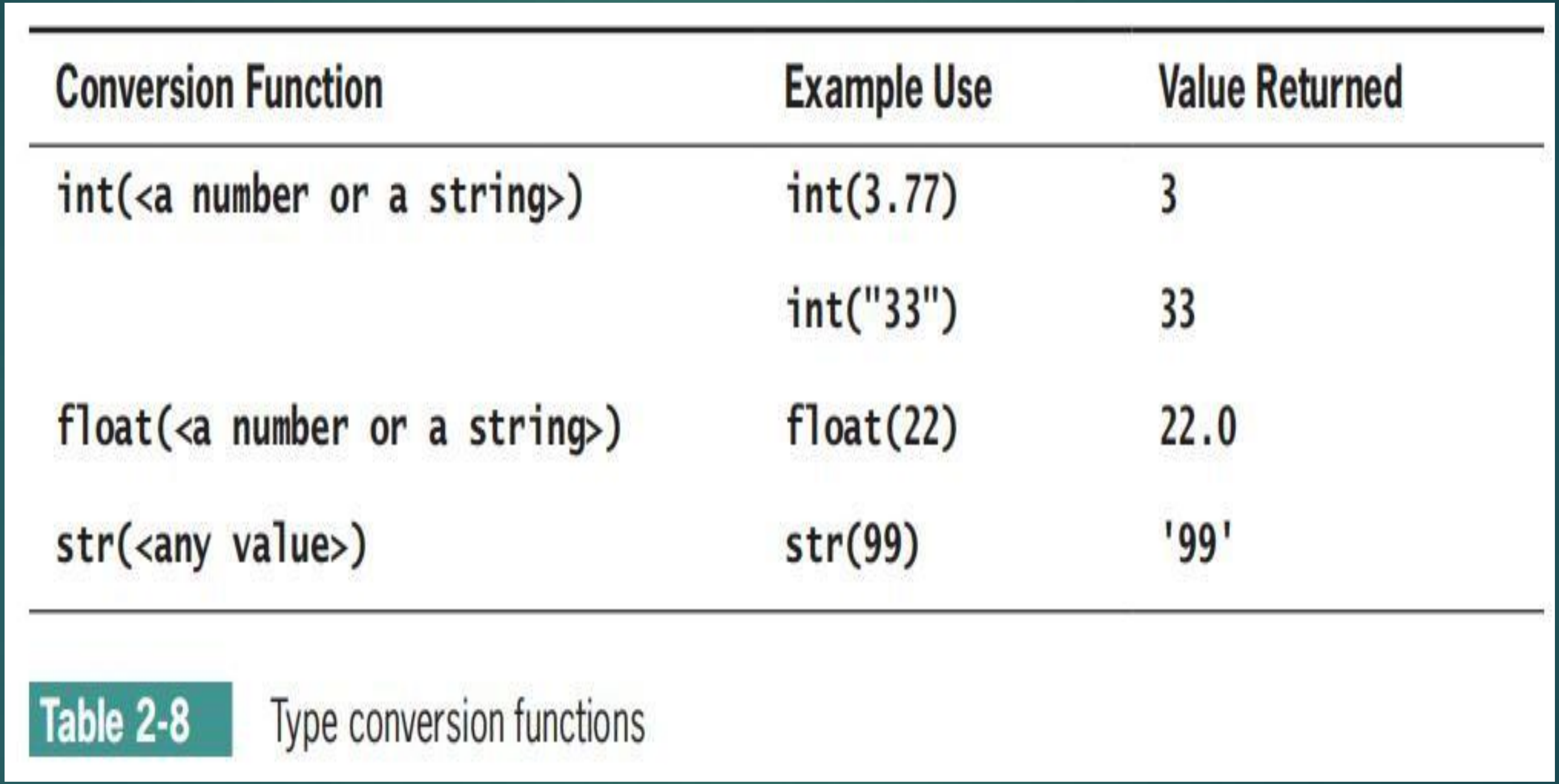
- *eg:*

```
>>> 3.14 * 3 ** 2
28.26
```

You must use a type conversion function when working with the input of numbers. A type conversion function is a function with the same name as the data type to which it converts.

Because the input function returns a string as its value, you must use the function int or float to convert the string to a number before performing arithmetic, as in the following example:

```
>>> radius = input("Enter the radius: ")
Enter the radius: 3.2
>>> radius
'3.2'
>>> float(radius)
3.2
>>> float(radius) ** 2 * 3.14
    32.153600000000004
```



Conversion Function	Example Use	Value Returned
`int(<a number or a string>)`	`int(3.77)`	`3`
	`int("33")`	`33`
`float(<a number or a string>)`	`float(22)`	`22.0`
`str(<any value>)`	`str(99)`	`'99'`

Table 2-8 Type conversion functions

Mixed-Mode Arithmetic and Type Conversions (4 of 4)

- Type conversion also occurs in the construction of strings from numbers and other strings

```
>>> profit = 1000.55
>>> print('$' + profit)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'float' objects
```

- Solution: use **str** function

```
>>> print('$' + str(profit))
$1000.55
```

- Python is a strongly **typed** programming language

Exercises

5) Let x = 8 and y = 2. Write the values of the following expressions:

a. x + y * 3
b. (x + y) * 3
c. x ** y
d. x % y
e. x / 12.0
f. x // 6

6) Let x = 4.66 Write the values of the following expressions:

a. round(x)
b. int(x)

Using Functions and Modules

- Python includes many useful functions, which are organized in libraries of code called modules.
- A function is a chunk of code that can be called by name to perform a task. Functions often require arguments, that is, specific data values, to perform their tasks.
- Names that refer to arguments are also known as parameters.
- The process of sending a result back to another part of a program is known as returning a value.

Ex. round(7.563,2)

return 7.56

abs(4-5)

returns 1

The above functions belons to__builtin__module

Some build-in functions in python

abs(x) returns the absolute value of x . abs(-45) will return 45
max(x,y,z) returns the maximum of x,y,z . max(10,20,30) will return 30
min(x,y,z) returns the minimum of x,y,z . min(10,20,30) will return 10
divmod(x,y) returns both the quotient and remainder . divmod(14,5) will return (2,4)
cmp(x,y) returns 0 if x==y , 1 if x>y and -1 if x<y
round(x,n) round x to n digits. round(3.14567,2) will return 3.15
range(start,stop,step) will return a list from start to stop-1 with an increment of step.
range(10) will return [0,1,2,…9] range(1,10,2) will return [1,3,5,7,9]
type(x) will return the type of the variable object x.
dir(x) will display the details of the object x.
len(x) will return the length of the object.
int(),float(),str(),bool(),chr(),long() these functions can be used for type conversions.
bin(x), oct(x), hex(x) these functions will convert the decimal number x into corresponding base.

What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

Example

Save this code in a file named mymodule.py

```
def greeting(name):
    print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule
mymodule.greeting("Jonathan")
```

The math Module

- The math module includes several functions that perform basic mathematical operations.
- This list of function names includes some familiar trigonometric functions as well as Python’s most exact estimates of the constants pi and e.

ex: 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc'..... etc.

✓ *syntax for using specific function from math module*

ex: math.pi, math.sqrt(2)

to import specific functions

ex: from math import pi, sqrt

✓ *to import all of the math module’s resources*

ex: from math import *

Program Format and Structure

- Start with an introductory comment stating the author’s name, the purpose of the program, and other relevant information. This information should be in the form of a docstring.
- Then, include statements that do the following:
  - ✓ Import any modules needed by the program.
  - ✓ Initialize important variables, suitably commented.
  - ✓ Prompt the user for input data and save the input data in variables.
  - ✓ Process the inputs to produce the results.

Control Statements

control statements—statements that allow the computer to select or repeat an action.

✓Selection structure : statements which determines whether other statements will be executed
e.g. if-else, switch-case

✓Iteration structure : statements which determines how many time to execute another statements.

e.g. for, while

Selection: if and if-else Statements

- if-else statement:

✓It is also called a two-way selection statement, because it directs the computer to make a choice between two alternative courses of action.

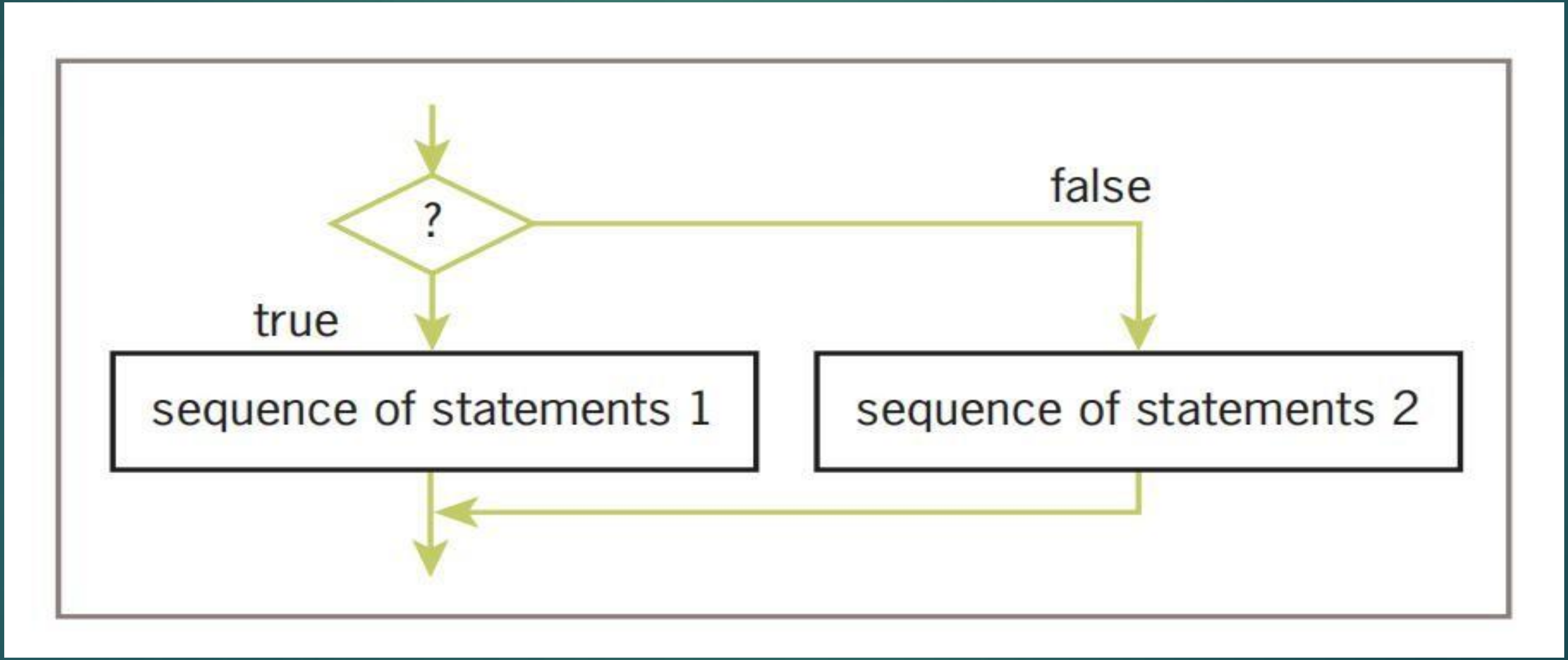
✓It is often used to check inputs for errors and to respond with error messages if necessary.

```
if <condition>:
          <sequence of statements–1>
else:
          <sequence of statements–2>
```

✓ The condition in the if-else statement must be a Boolean expression—that is, an expression that evaluates to either true or false.

if-else Statements

e.g., prints the maximum and minimum of two input numbers.



Selection: if and if-else Statements

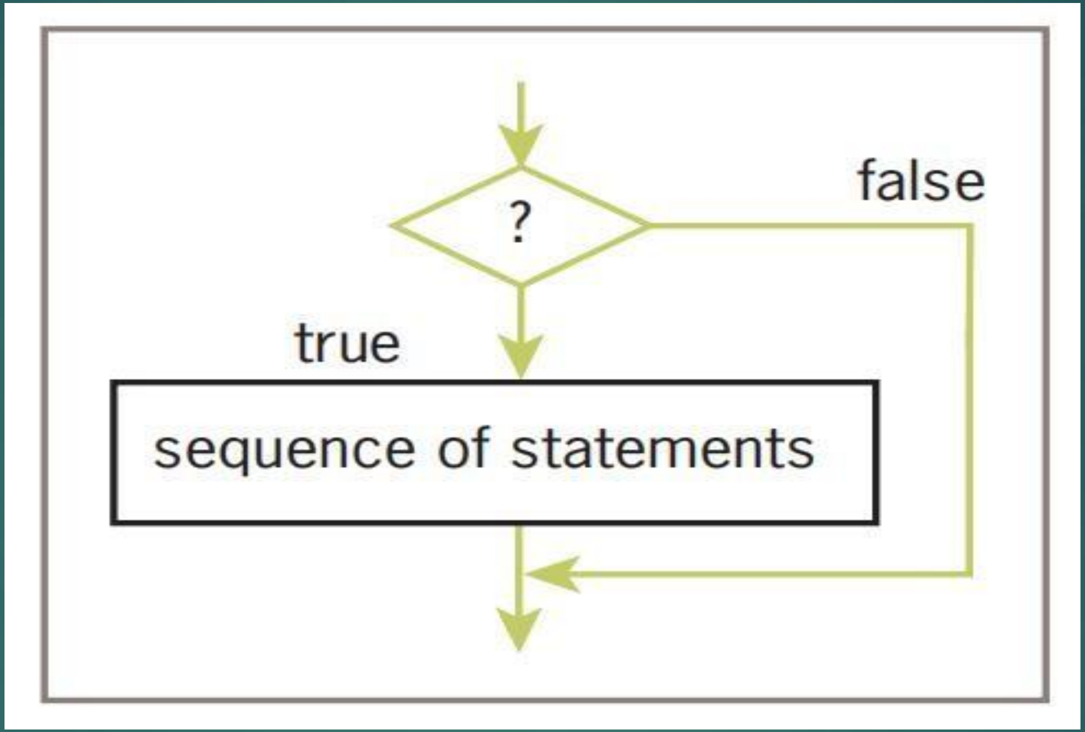
- if statement:

✓The simplest form of selection is the if statement. This type of control statement is also called a one-way selection statement.

✓it consists of a condition and just a single sequence of statements. If the condition is True, the sequence of statements is run. Otherwise, control proceeds to the next statement following the entire selection statement.

```
if <condition>:
   <sequence of statements>
```

if - Statements

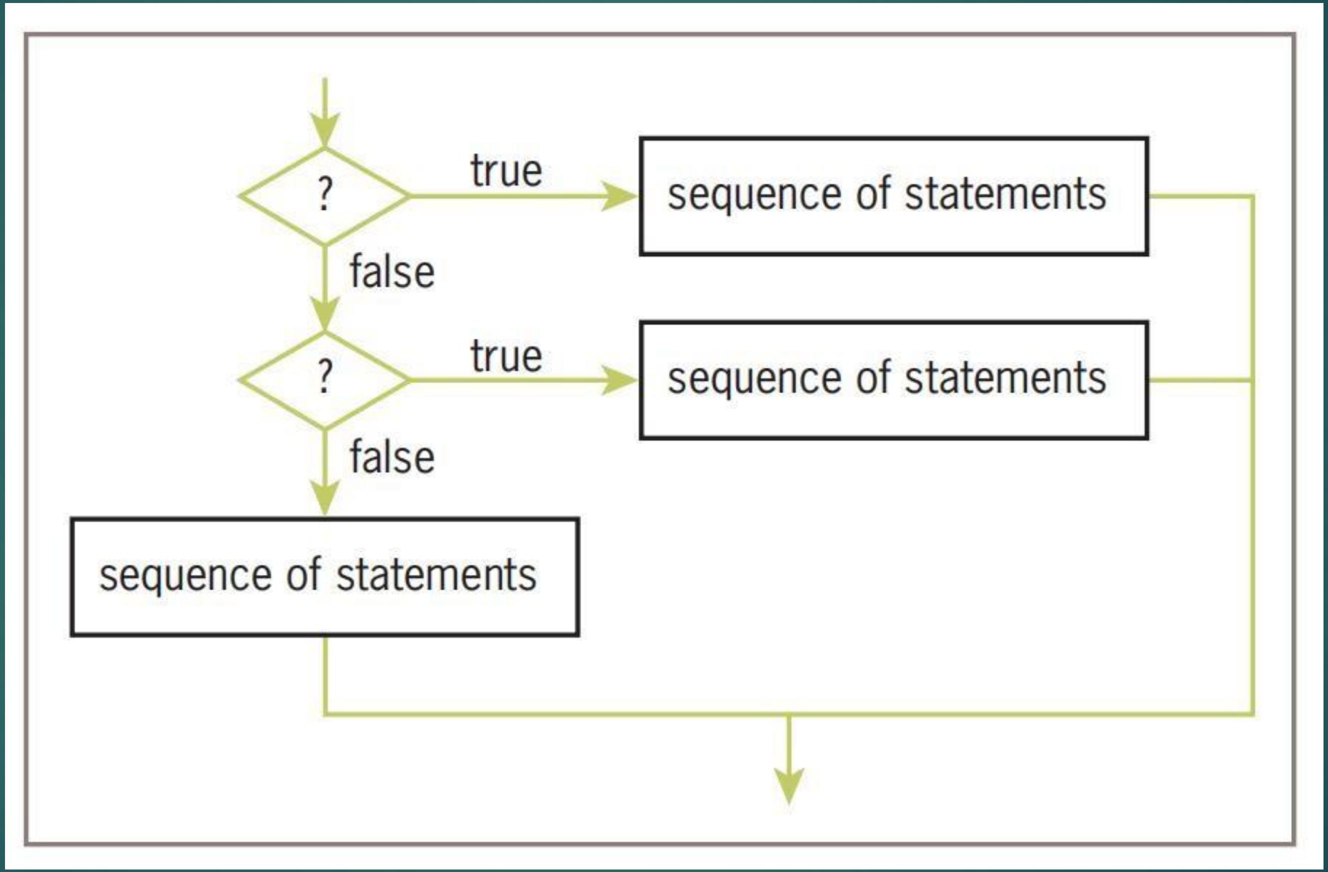


```
if x < 0:
        x = –x
```

The process of testing several conditions and responding accordingly can be described in code by a multi-way selection statement.

```
if <condition-1>:
 <sequence of statements-1>
.
.
.
elif <condition-n>:
 <sequence of statements-n>
else:
 <default sequence of statements>
```

Multiway if statement

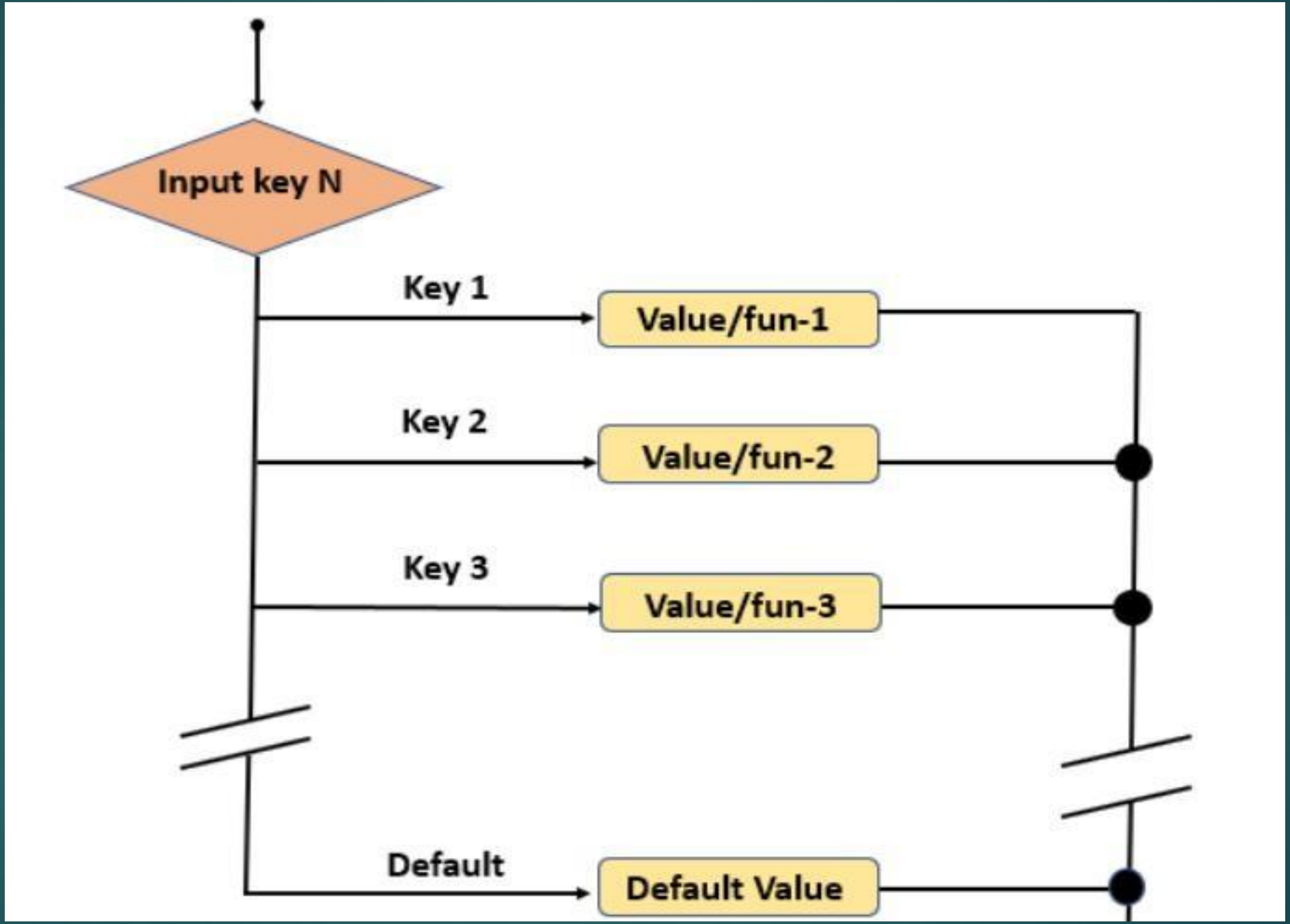


Introduction to Python Switch Statement

- A switch statement is a very useful and powerful programming feature. It is an alternate to if-else-if ladder statement and provides better performance and more manageable code than an if-else-if ladder statement.

- Python language does not provide any inbuilt switch statements. We can implement this feature with the same flow and functionality but with different syntax and implementation using Python Dictionary.

Flow Chart


Input key N
Key 1
Value/fun-1
Key 2
Value/fun-2
Key 3
Value/fun-3
Default
Default Value

Syntax of Switch Statement

1. Switch in Other Languages (c, Java,..)

```
switch(N)
{
case 1: Statement if N = 1;
break;
case 2: Statement if N = 2;
break;
::
case n: Statement if N = n;
break;
default: Statement if N doesn't match any
}
```

2. Switch Implementation in Python

```
switcher=
{
key_1:
value_1/method_1(),
key_2:
value_2/method_2(),
key_3:
value_3/method_3(),
::
key_n: value_n/method_n(),
}
key = N
value = switcher.get(key, "default")
```

Implementation of Switch statement in Python

```
def get_week_day(argument):
switcher = {
0: "Sunday",
1: "Monday",
2: "Tuesday",
3: "Wednesday",
4: "Thursday",
5: "Friday",
6: "Saturday"
}
return switcher.get(argument, "Invalid day")
print (get_week_day(6))
print (get_week_day(8))

print (get_week_day(0))
```

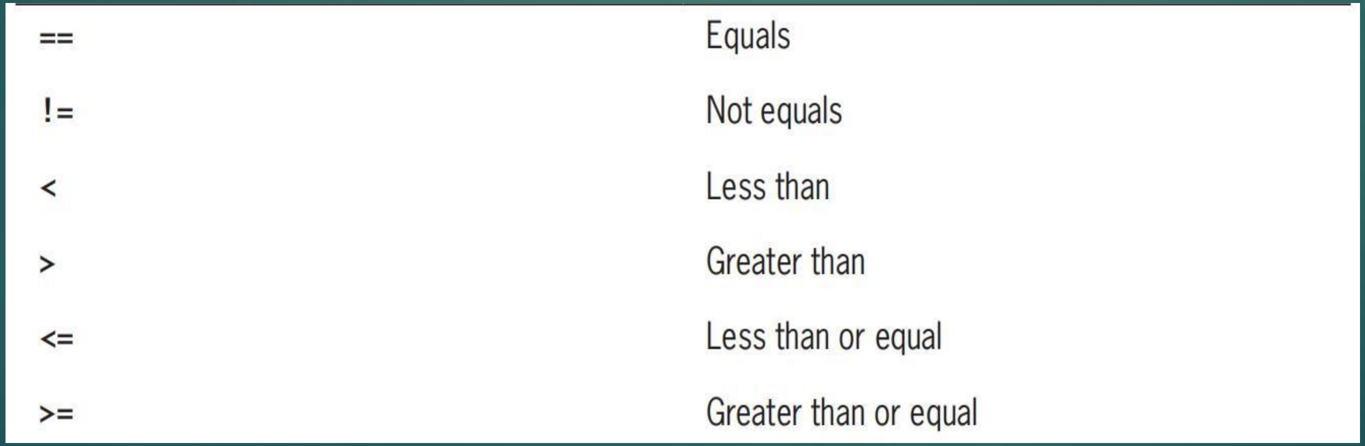
```
def vowel(num):
        switch={
        1:'a',
        2:'e',
        3:'i',
        4:'o',
        5:'u'
        }
 return switch.get(num,"Invalid input")
vowel(3)
vowel(0)
```

Here the output will be

```
‘i’
‘Invalid input’
```

The Boolean Type, Comparisons, and Boolean Expressions

Boolean Data Type: The Boolean data type consists of only two data values—true and false. Simple Boolean expressions consist of the Boolean values True or False, variables bound to those values, function calls that return Boolean values, or comparisons.



`==`	Equals
`!=`	Not equals
`<`	Less than
`>`	Greater than
`<=`	Less than or equal
`>=`	Greater than or equal

Logical Operators and Compound Boolean Expressions

```
number = int(input("Enter the numeric grade: "))
if number > 100:
        print("Error: grade must be between 100 and 0")
elif number < 0:
        print("Error: grade must be between 100 and 0")
else:
 # The code to compute and print the result goes here
```

- The two conditions can be combined in a Boolean expression that uses the logical operator **or**. The resulting expression is compound Boolean expression

```
number = int(input("Enter the numeric grade: "))
if number > 100 or number < 0:
        print("Error: grade must be between 100 and 0")
else:
 # The code to compute and print the result goes here
```

- Another way to describe this situation using and operator is

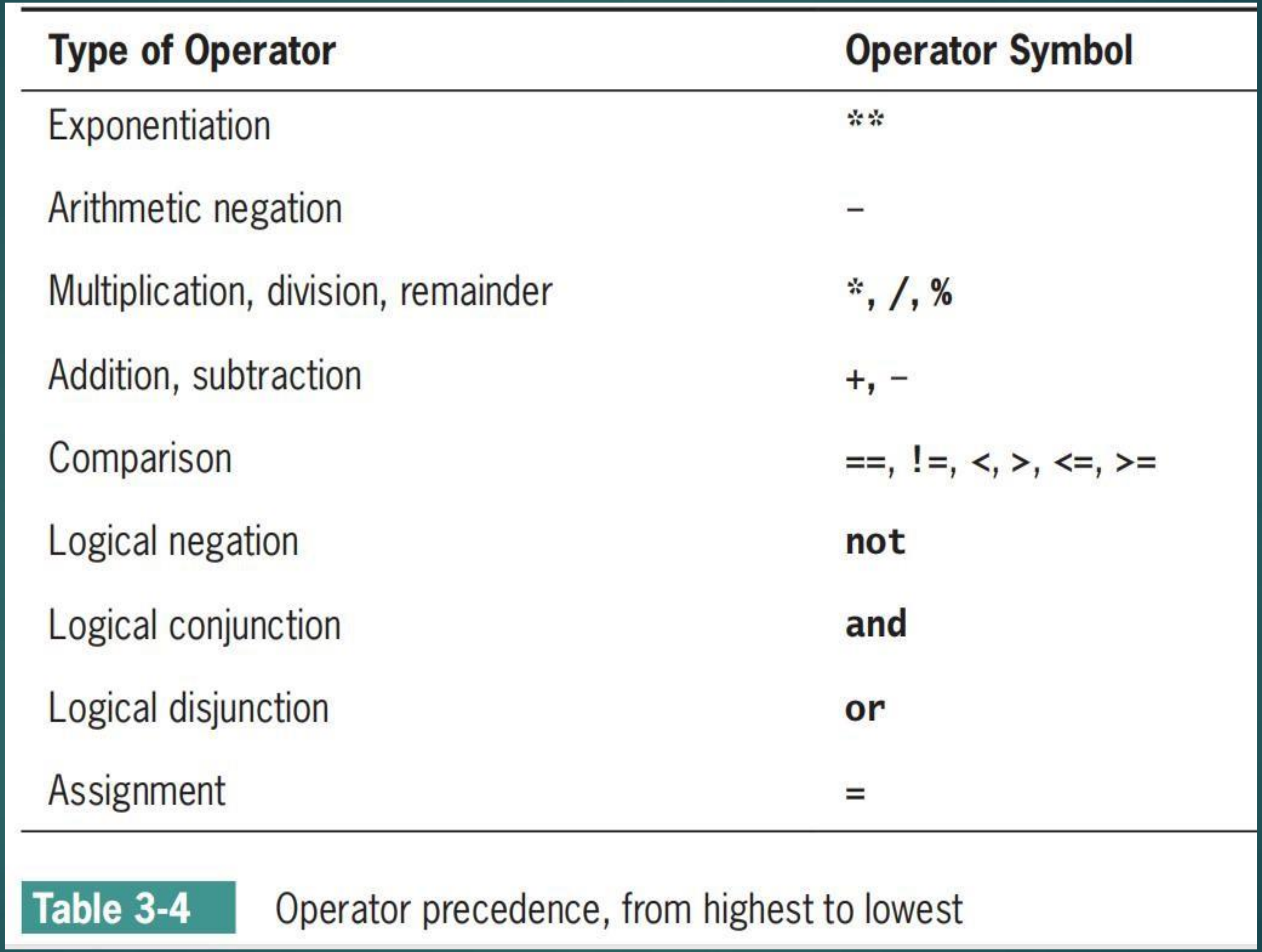
```
number = int(input("Enter the numeric grade: "))
if number >= 0 and number <= 100:
    # The code to compute and print the result goes here
else:
    print("Error: grade must be between 100 and 0")
```

Precedence of Logical Operators

```
>>> A = True
>>> B = False
>>> A and B
      False
>>> A or B
     True
>>> not A
    False
```

The logical operators are evaluated after comparisons but before the assignment operator. The not operator has a higher precedence than the and operator, which has a higher precedence than the or operator.

Thus, in our example, not A and B evaluates to False, whereas not (A and B) evaluates to True.



Type of Operator	Operator Symbol
Exponentiation	`**`
Arithmetic negation	`-`
Multiplication, division, remainder	`*, /, %`
Addition, subtraction	`+, -`
Comparison	`==, !=, <, >, <=, >=`
Logical negation	**not**
Logical conjunction	**and**
Logical disjunction	**or**
Assignment	`=`

Table 3-4 Operator precedence, from highest to lowest

Lazy Evaluation – Advantages

- It allows the language runtime to discard sub-expressions that are not directly linked to the final result of the expression.
- It reduces the time complexity of an algorithm by discarding the temporary computations and conditionals.
- It allows the programmer to access components of data structures out-of-order after initializing them, as long as they are free from any circular dependencies.
- It is best suited for loading data which will be infrequently accessed.

Lazy Evaluation – Drawbacks

- It forces the language runtime to hold the evaluation of sub-expressions until it is required in the final result by creating thunks (delayed objects).
- Sometimes it increases space complexity of an algorithm.
- It is very difficult to find its performance because it contains thunks of expressions before their execution.

Lazy Evaluation using Python

The range method in Python follows the concept of Lazy Evaluation. It saves the execution time for larger ranges and we never require all the values at a time, so it saves memory consumption as well. Take a look at the following example.

```
r = range(10)

print(r)

range(0, 10)

print(r[3])
```

It will produce the following output −

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

3
```

Short-Circuit Evaluation

The Python virtual machine sometimes knows the value of a Boolean expression before it has evaluated all of its operands.

For instance, in the expression A and B, if A is false, there is no need to evaluate B. Likewise, in the expression A or B, if A is true, there is no need to evaluate B.

This approach, in which evaluation stops as soon as possible, is called short-circuit evaluation.

```
count = int(input("Enter the count: "))
sum = int(input("Enter the sum: "))
if count > 0 and sum // count > 10:
    print("average > 10")
else:
    print("count = 0 or average <= 10")
```

Exercises:

- 1. Assume that x is 3 and y is 5. Write the values of the following expressions:
  a) x == y
  b) x > y – 3
  c) x <= y – 2
  d) x == y or x > 2
  e) x != 6 and y > 10
  f) x > 0 and x < 100

2. Assume that x refers to a number. Write a code segment that prints the number's absolute value without using Python's abs function.

Definite Iteration: The for Loop

Loops - They are the control statements with repetition statements, also known as loops, which repeat an action. Each repetition of the action is known as a pass or an iteration.

There are two types of loops—

- ✓ those that repeat an action a predefined number of times (definite iteration)
- ✓ those that perform the action until the program determines that it needs to stop (indefinite iteration).

Syntax:

```
for <variable> in range(<an integer expression>):
 <statement-1>
 .
 .
 <statement-n>
```

- The first line of code in a loop is sometimes called the loop header.
- The integer value or expression in the range() tells the loop how many times to call the below statements.
- The loop body comprises the statements in the remaining lines of code, below the header. These statements are executed in sequence on each pass through the loop.

- Implementation of exponentiation operator using for loop

```
#computation of 2³
>>> number = 2
>>> exponent = 3
>>> product = 1
>>> for eachPass in range(exponent):
 product = product * number
 print(product, end = " ")
2 4 8
>>> product
8
```

Count-Controlled Loops

- ✓Loops that count through a range of numbers are also called count- controlled loops.
- ✓ It counts from 0 to the value of the header's integer expression minus 1.
- ✓ On each pass through the loop, the header's variable is bound to the current value of this count.

e.g.,

```
>>> for count in
    range(4):
    print(count, end =
    " ")

    0 1 2 3
```

- ✓To count from an explicit lower bound, the programmer can supply a second integer expression in the loop header. When two arguments are supplied to range, the count ranges from the first argument to the second argument minus 1.

Syntax:

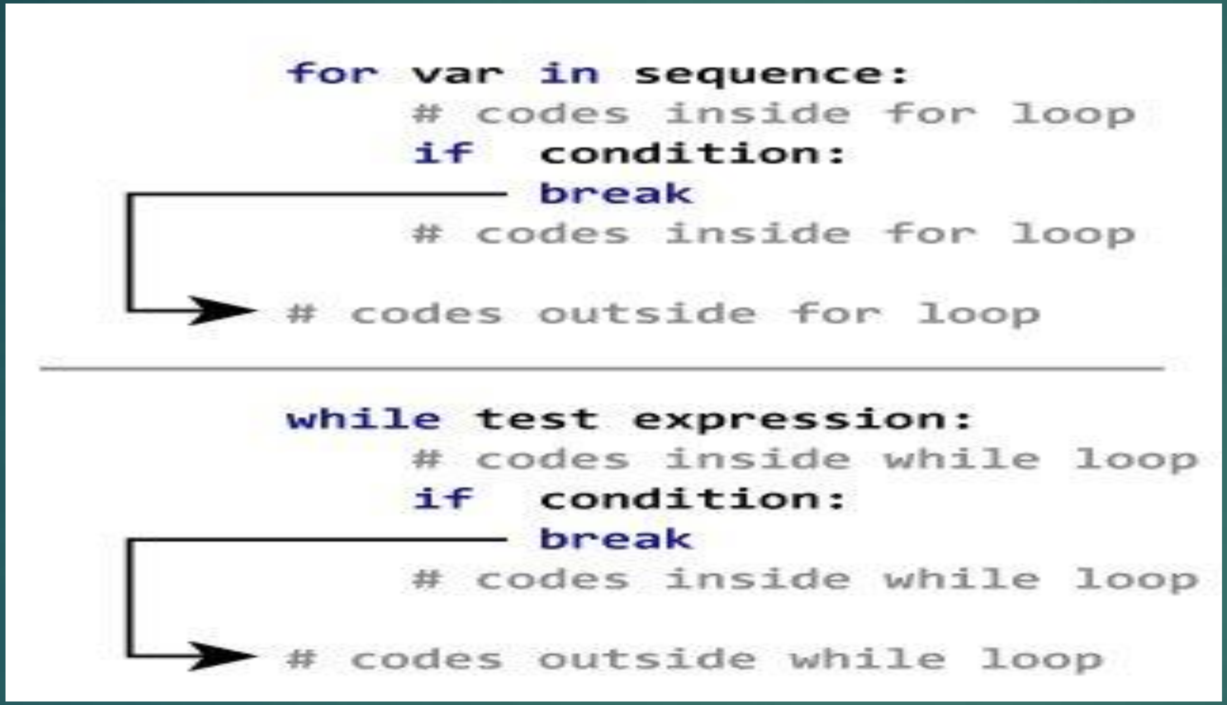
```
for <variable> in range(<lower bound>, <upper bound + 1>):
  <loop body>
```

```
# method - 1 :  computation of factorial of 4
>>> product = 1
>>> for count in range(4):
product = product * (count + 1)
>>> product
24
```

```
# method - 2 :  computation of factorial of 4
>>> product = 1
>>> for count in range(1, 5):
 product = product * count
>>> product
24
```

# program to find the sum of first 10 integer numbers:

```
>>> lower = int(input("Enter the lower bound: "))
Enter the lower bound: 1
>>> upper = int(input("Enter the upper bound: "))
Enter the upper bound: 10
>>>
>>>theSum = 0
>>>for number in range(lower, upper+1):
      theSum = theSum + number
>>>print(theSum)


55
```



```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

Augmented Assignment

- The assignment symbol can be combined with the arithmetic and concatenation operators to provide augmented assignment operations.

```
✓a = 17
✓s = "hi"
✓a += 3             # Equivalent to a = a + 3
✓a -= 3             # Equivalent to a = a - 3
✓a *= 3             # Equivalent to a = a * 3
✓a /= 3             # Equivalent to a = a / 3
✓a %= 3             # Equivalent to a = a % 3
✓s += " there"      # Equivalent to s = s + " there"
```

All these examples have the format

<variable> <operator>= <expression>

Loop Errors: Off-by-One Error

- Once we get the syntax correct, we need to be concerned about only one other possible error: The loop fails to perform the expected number of iterations. Because this number is typically off by one, the error is called an off-by-one error.

- For the most part, off-by-one errors result when the programmer incorrectly specifies the upper bound of the loop.

- # Count from 1 through 4, we think
- >>> for count in range(1,4):
- print(count)
-
- 1 2 3

Traversing the Contents of a Data Sequence

- The for loop itself visits each number in a sequence of numbers generated by the range function.
- The sequence of numbers generated by the function range is fed to Python's list function, which returns a special type of sequence called a list.

```
>>> list(range(4))
[0, 1, 2, 3]
>>> list(range(1, 5))
[1, 2, 3, 4]
```

Strings are also sequences of characters. The values contained in any sequence can be visited by running a for loop, as follows:

```
for <variable> in <sequence>:
    <do something with variable>
```

Specifying the Steps in the Range

- In some programs we might want a loop to skip some numbers.
- A variant of Python's range function expects a third argument that allows you to nicely skip some numbers. The third argument specifies a step value, or the interval between the numbers used in the range.

```
>>> list(range(1, 6, 1)) # Same as using two arguments
[1, 2, 3, 4, 5]
>>> list(range(1, 6, 2)) # Use every other number
[1, 3, 5]
>>> list(range(1, 6, 3)) # Use every third number
[1, 4]
```

Suppose you had to compute the sum of the even numbers between 1 and 10.

```
>>> theSum = 0
>>> for count in range(2, 11, 2):
 theSum += count
>>>
theSum  30
```

Loops that Count Down

When the step argument is a negative number, the range function generates a sequence of numbers from the first argument down to the second argument plus 1. Thus, in this case, the first argument should express the upper bound, and the second argument should express the lower bound minus 1.

Exercises

1. Write the outputs of the following loops:

a. for count in range(5):

```
print(count + 1, end = " ")
```

b. for count in range(1, 4):

```
print(count, end = " ")
```

c. for count in range(1, 6, 2):

```
print(count, end = " ")
```

d. for count in range(6, 1, –1):

```
print(count, end = " ")
```

Formatting Text for Output

To maintain the margins between columns of data, left-justification requires the addition of spaces to the right of the datum, whereas right-justification requires adding spaces to the left of the datum. A column of data is centered if there are an equal number of spaces on either side of the data within that column.

The total number of data characters and additional spaces for a given datum in a formatted string is called its field width.

how to right-justify and left-justify the string "four" within a field width of 6:

```
>>> "%6s" % "four"              # Right justify
'  four'
>>> "%-6s" % "four"             # Left justify
'four  '
```

The simplest form of this operation is the following:
<format string> % <datum>

- ✓The format string can contain string data and other information about the format of the datum.
- ✓When the field width is positive, the datum is right-justified; when the field width is negative, you get left-justification.
- ✓If the field width is less than or equal to the datum's print length in characters, no justification is added.

- ✓ To format a sequence of data values,

<format string> % (<datum–1>, ..., <datum–n>)

The format information for a data value of type float has the form

%<field width> . <precision>f

```
>>> salary = 100.00
>>> print("Your salary is $" +
str(salary))  Your salary is $100.0

>>> print("Your salary is $%0.2f" %
salary)  Your salary is $100.00
```

To use a field width of 6 and a precision of 3 to format the float value 3.14:

```
>>> "%6.3f" % 3.14
' 3.140'
```

Exercises:

- 1. Assume that the variable amount refers to 24.325. Write the outputs of the following statements:

  a. print("Your salary is $%0.2f" % amount)

  b. print("The area is %0.1f" % amount)

2. Write a code segment that displays the values of the integers x, y, and z on a single line, such that each value is right-justified with a field width of 6.

3. Write a format operation that builds a string for the float variable amount that has exactly two digits of precision and a field width of zero.

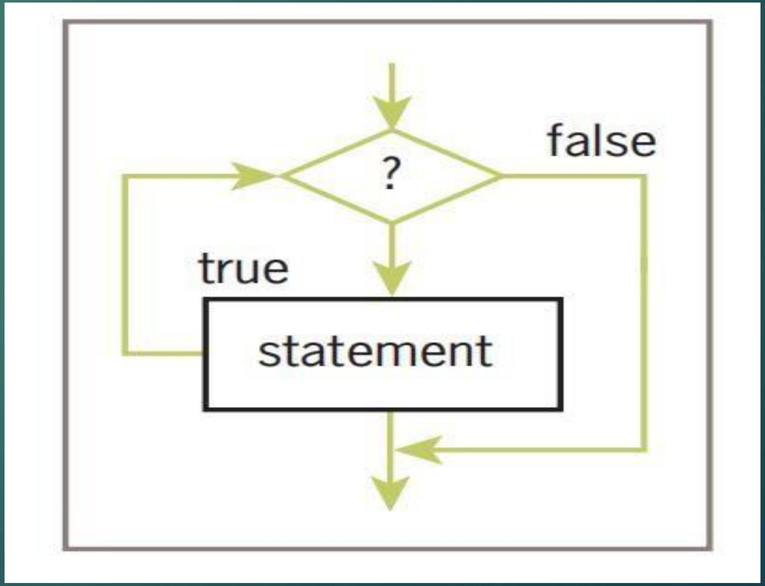
4. Write a loop that outputs the numbers in a list named salaries. The outputs should be formatted in a column that is right-justified, with a field width of 12 and a precision of 2.

Conditional Iteration: The while Loop

In many situations, however, the number of iterations in a loop is unpredictable. The loop eventually completes its work, but only when a condition changes. The process continues to repeat as long as a condition remains true. This type of process is called conditional iteration, meaning that the process continues to repeat as long as a condition remains true.

Syntax:

```
while <condition>:
 <sequence of statements>
```



The Structure and Behavior of the Loop

- The form of this statement is almost identical to that of the one-way selection statement.
- The use of the reserved word while instead of if indicates that the sequence of statements might be executed many times, as long as the condition remains true.
- Clearly, something eventually has to happen within the body of the loop to make the loop’s continuation condition become false. Otherwise, the loop will continue forever, an error known as an infinite loop.
- At least one statement in the body of the loop must update a variable that affects the value of the condition.

Consider the pseudocode given below:

```
set the sum to 0.0
input a string
while the string is not the empty string
    convert the string to a float
    add the float to the sum
    input a string
print the sum
```

- The first input statement initializes a variable to a value that the loop condition can test. This variable is also called the loop control variable.
- The second input statement obtains the other input values, including one that will terminate the loop.

```
theSum = 0.0
data = input("Enter a number or just enter to quit: ")
while data != "":
        number = float(data)
        theSum += number
       data = input("Enter a number or just enter to quit: ")
print("The sum is", theSum)
```

Output:

```
Enter a number or just enter to quit: 3
Enter a number or just enter to quit: 4
Enter a number or just enter to quit: 5
Enter a number or just enter to quit:
The sum is 12.0
```

The while loop is also called an entry-control loop, because its condition is tested at the top of the loop. This implies that the statements within the loop can execute zero or more times.

Count Control with a while Loop

```
# Summation with a for loop
theSum = 0
for count in range(1, 11):
    theSum += count
print(theSum)
```

```
# Summation with a while loop
theSum = 0
count = 1
while count <= 10:
    theSum += count
    count += 1
print(theSum)
```

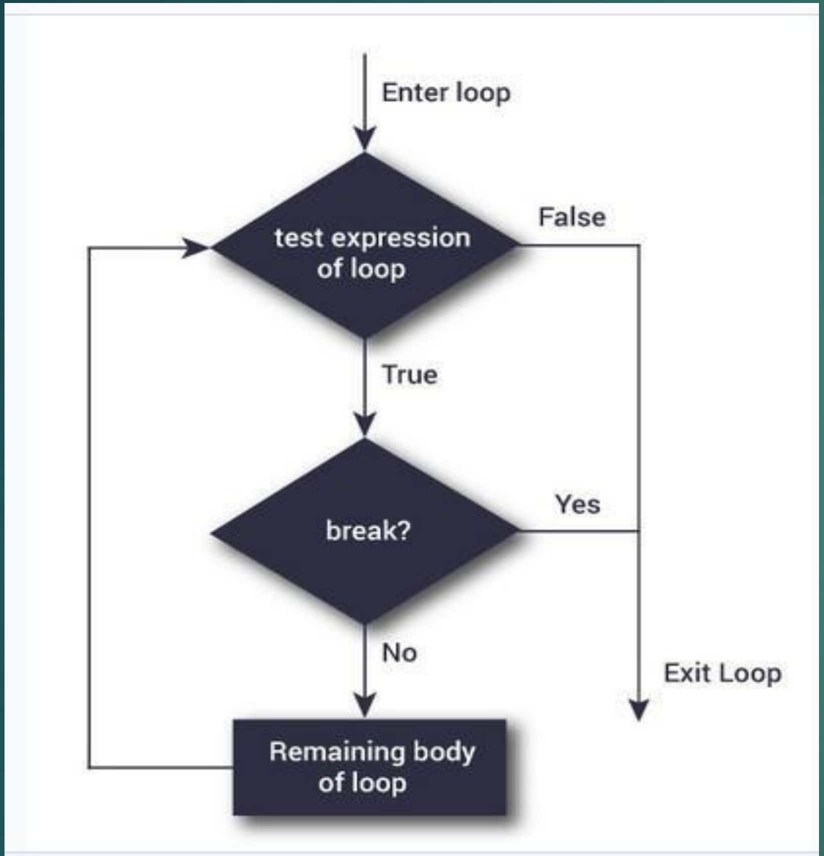
It includes a Boolean expression and two extra statements that refer to the count variable. This loop control variable must be explicitly initialized before the loop header and incremented in the loop body. *

The while True Loop and the break Statement

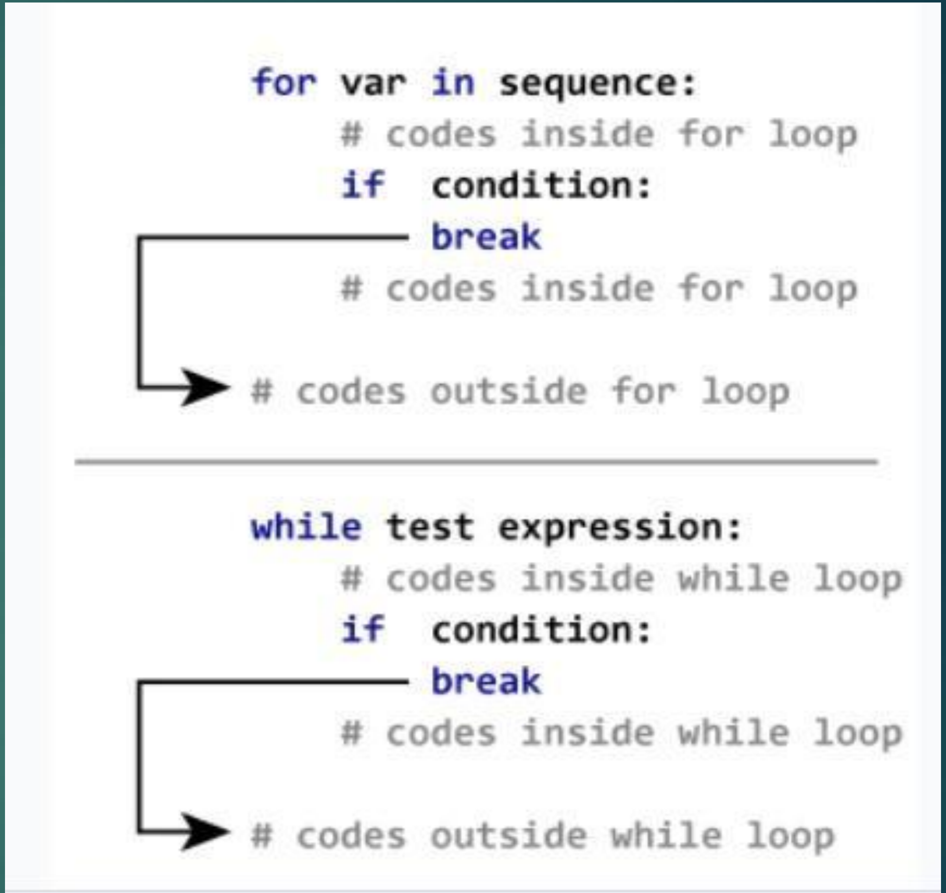
```
#Computation of sum using break within while loop

theSum = 0.0
while True:
    data = input("Enter a number or just enter to quit: ")
    if data == "":
        break
    number = float(data)
    theSum += number
print("The sum is", theSum)
```

Within this body, the input datum is received. It is then tested for the loop's termination condition in a one-way selection statement. If the user wants to quit, the input will equal the empty string, and the break statement will cause an exit from the loop. Otherwise, control continues beyond the selection statement to the next two statements that process the input.

The “break” Statement



Flowchart of break statement in Python

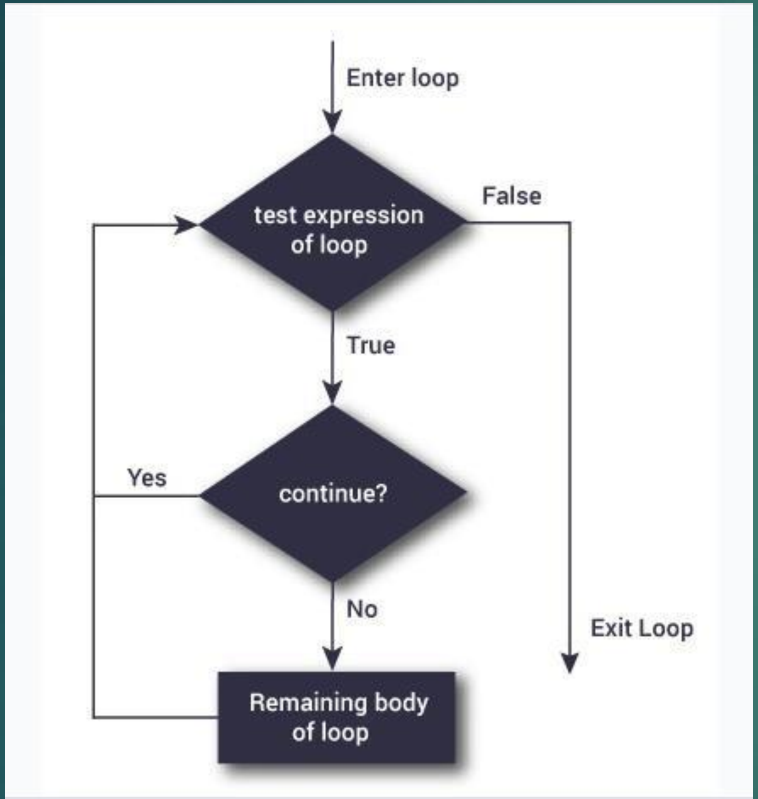


```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

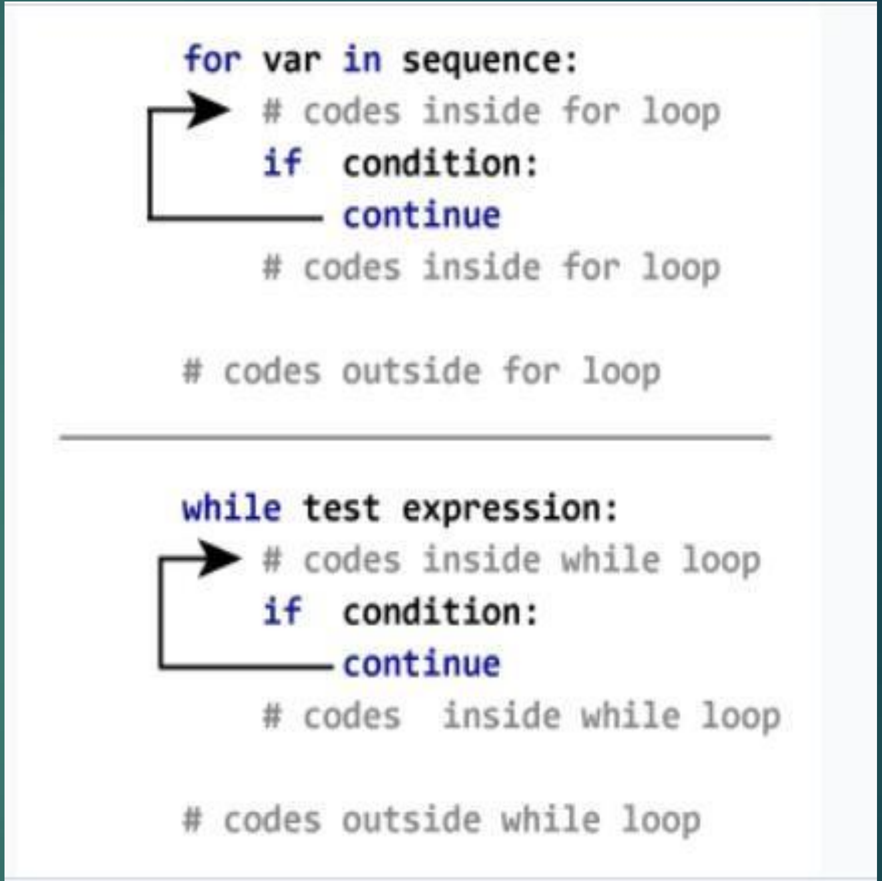
```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

Working of the break statement

The “continue” Statement



Flowchart of continue statement in Python



```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

How continue statement works in python

Exercises

1. Translate the following for loops to equivalent while loops:

```
a. for count in range(100):
      print(count)
b. for count in range(1, 101):
      print(count)
c. for count in range(100, 0, –1):
      print(count)
```

2.The factorial of an integer N is the product of the integers between 1 and N, inclusive. Write a while loop that computes the factorial of a given integer N.

Exercises

3.Write a program that accepts the lengths of three sides of a triangle as inputs. The program output should indicate whether or not the triangle is an equilateral triangle.

4. Write a program that accepts the lengths of three sides of a triangle as inputs. The program output should indicate whether or not the triangle is a right triangle. Recall from the Pythagorean theorem that in a right triangle, the square of one side equals the sum of the squares of the other two sides.