Python Programming

Using Problem Solving Approach

Prepared By:
Prof. Prachi Pancholi

CHAPTER 3

Basics of Python Programming

Features of Python

- Simple
- Easy to Learn
- Versatile
- Free and Open Source
- High-level Language
- Interactive
- Portable
- Object Oriented
- Interpreted
- Dynamic
- Extensible
- Embeddable
- Extensive
- Easy maintenance
- Secure
- Robust
- Multi-threaded
- Garbage Collection

Limitations of Python

Parallel processing can be done in Python but not as elegantly as done in some other languages (like JavaScript and Go Lang).

- Being an interpreted language, Python is slow as compared to C/C++. Python is not a very good choice for those developing a high-graphic 3d game that takes up a lot of CPU.
- As compared to other languages, Python is evolving continuously and there is little substantial documentation available for the language.
- As of now, there are few users of Python as compared to those using C, C++ or Java.
- It lacks true multiprocessor support.
- It has very limited commercial support point.
- Python is slower than C or C++ when it comes to computation heavy tasks and desktop applications.
- It is difficult to pack up a big Python application into a single executable file. This makes it difficult to distribute Python to non-technical.

Applications of Python

- **Embedded scripting language:** Python is used as an embedded scripting language for various testing/ building/ deployment/ monitoring frameworks, scientific apps, and quick scripts.
- **3D Software:** 3D software like Maya uses Python for automating small user tasks, or for doing more complex integration such as talking to databases and asset management systems.
- **Web development:** Python is an easily extensible language that provides good integration with database and other web standards.

GUI-based desktop applications: Simple syntax, modular architecture, rich text processing tools and the ability to work on multiple operating systems makes Python a preferred choice for developing desktop-based applications.

- ***Image processing and graphic design applications:*** Python is used to make 2D imaging software such as Inkscape, GIMP, Paint Shop Pro and Scribus. It is also used to make 3D animation packages, like Blender, 3ds Max, Cinema 4D, Houdini, Lightwave and Maya.

Applications of Python

- ***Scientific and computational applications:*** **Features like high speed, productivity and availability of tools, such as Scientific Python and Numeric Python, have made Python a preferred language to perform computation and processing of scientific data. 3D modeling software, such as FreeCAD, and finite element method software, like Abaqus, are coded in Python.**

Games: **Python has various modules, libraries, and platforms that support development of games. Games like Civilization-IV, Disney's Toontown Online, Vega Strike, etc. are coded using Python.**

- ***Enterprise and business applications:*** **Simple and reliable syntax, modules and libraries, extensibility, scalability together make Python a suitable coding language for customizing larger applications. For example, Reddit which was originally written in Common Lips, was rewritten in Python in 2005. A large part of Youtube code is also written in Python.**
- ***Operating Systems:*** **Python forms an integral part of Linux distributions.**

Writing and Executing First Python Program

Step 1: Open an editor.

Step 2: Write the instructions

Step 3: Save it as a file with the filename having the extension .py.

Step 4: Run the interpreter with the command python program_name.py or use IDLE to run the programs.

To execute the program at the *command prompt*, simply change your working directory to C:\Python34 (or move to the directory where you have saved Python) then type python program_name.py.

If you want to execute the program in Python shell, then just press F5 key or click on Run Menu and then select Run Module.

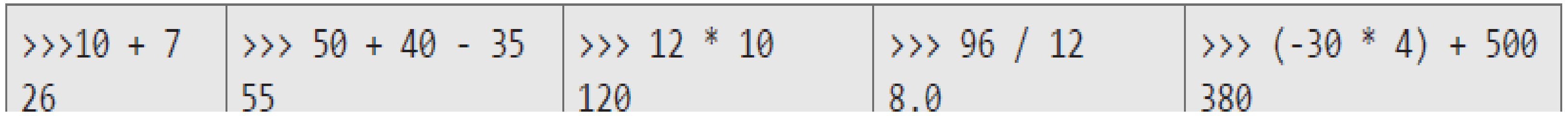
Literal Constants

The value of a literal constant can be used directly in programs. For example, 7, 3.9, 'A', and "Hello" are literal constants.

Numbers refers to a numeric value.You can use four types of numbers in Python program- integers, long integers, floating point and complex numbers.

- **Numbers like 5 or other whole numbers are referred to as *integers*. Bigger whole numbers are called *long integers*. For example, 535633629843L is a long integer.**
- **Numbers like are 3.23 and 91.5E-2 are termed as *floating point numbers*.**
- **Numbers of a + bi form (like -3 + 7i) are *complex numbers*.**

Examples:



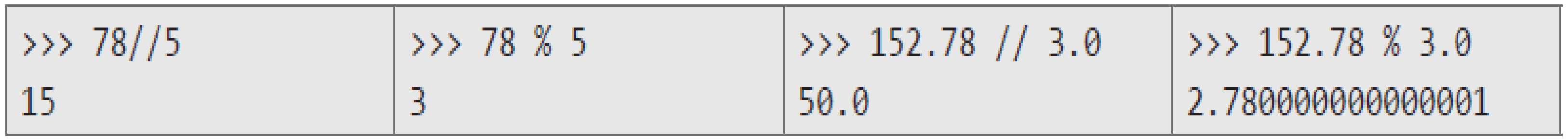
```
>>>10 + 7
26
```

```
>>> 50 + 40 - 35
55
```

```
>>> 12 * 10
120
```

```
>>> 96 / 12
8.0
```

```
>>> (-30 * 4) + 500
380
```

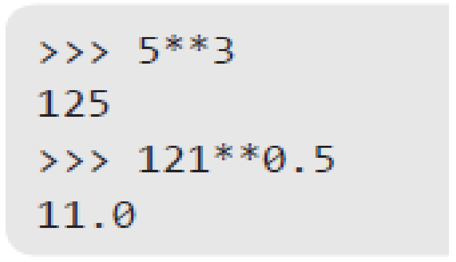


```
>>> 78//5
15
```

```
>>> 78 % 5
3
```

```
>>> 152.78 // 3.0
50.0
```

```
>>> 152.78 % 3.0
2.7800000000000001
```



```
>>> 5**3
125
>>> 121**0.5
11.0
```

Literal Constants

Strings

A *string* is a group of characters.

- ***Using Single Quotes ('):*** **For example, a string can be written as 'HELLO'.**
- ***Using Double Quotes ("):*** **Strings in double quotes are exactly same as those in single quotes. Therefore, 'HELLO' is same as "HELLO".**
- ***Using Triple Quotes (''' '''):*** **You can specify multi-line strings using triple quotes. You can use as many single quotes and double quotes as you want in a string within triple quotes.**

Examples:

```
>>> 'Hello'
'Hello'
```

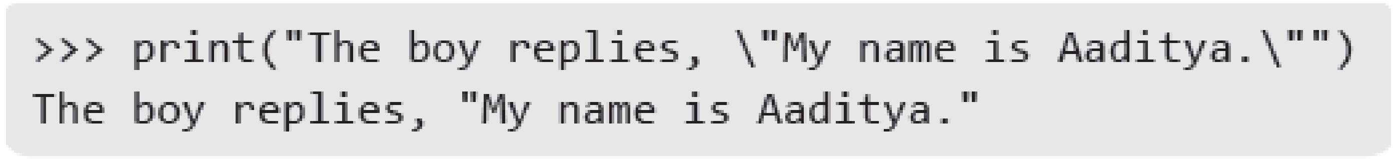
```
>>> "HELLO"
'HELLO'
```

```
>>> '''HELLO'''
'HELLO'
```

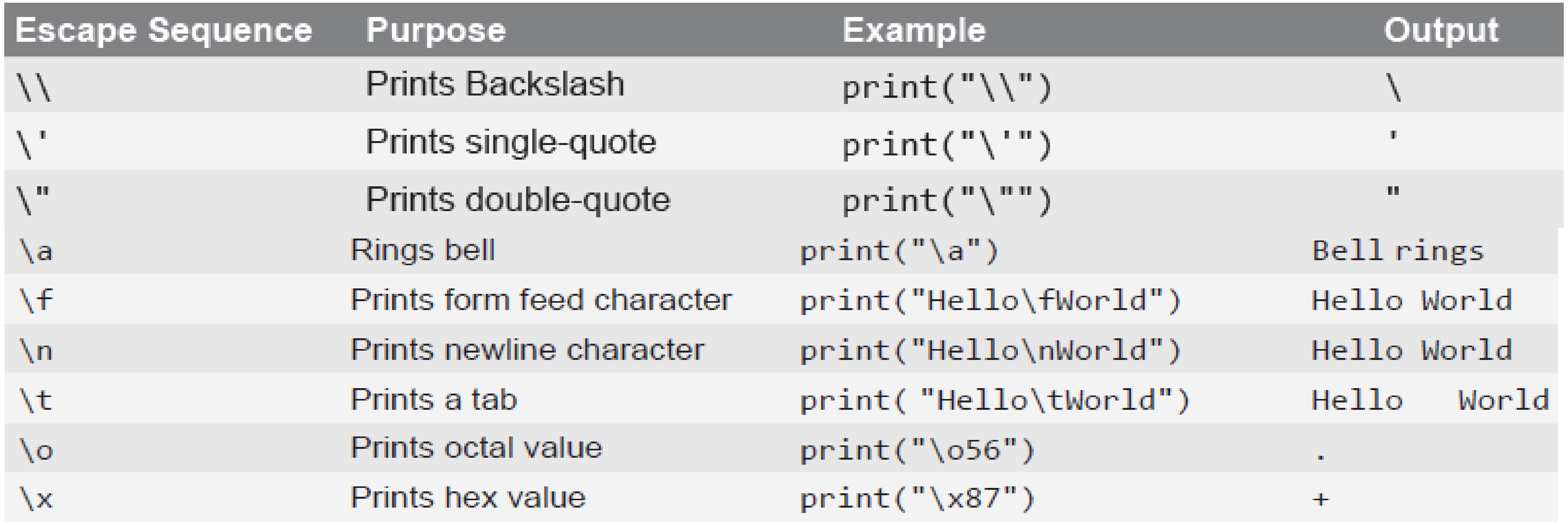
Escape Sequences

Some characters (like ", \) cannot be directly included in a string. Such characters must be escaped by placing a backslash before them.

Example:



```
>>> print("The boy replies, \"My name is Aaditya.\"")
The boy replies, "My name is Aaditya."
```

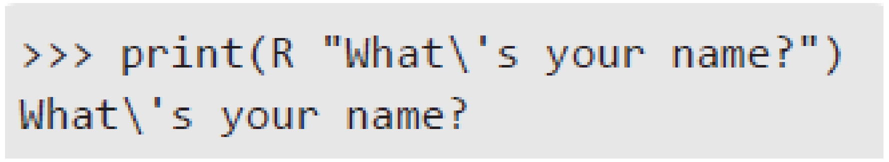


Escape Sequence	Purpose	Example	Output
\\	Prints Backslash	print("\\")	\
\'	Prints single-quote	print("\'")	'
\"	Prints double-quote	print("\"")	"
\a	Rings bell	print("\a")	Bell rings
\f	Prints form feed character	print("Hello\fWorld")	Hello World
\n	Prints newline character	print("Hello\nWorld")	Hello World
\t	Prints a tab	print("Hello\tWorld")	Hello World
\o	Prints octal value	print("\o56")	.
\x	Prints hex value	print("\x87")	+

Raw Strings

If you want to specify a string that should not handle any escape sequences and want to display exactly as specified then you need to specify that string as a *raw string*. A raw string is specified by prefixing r or R to the string.

Example:



```
>>> print(R "What\'s your name?")
What\'s your name?
```

Variables and Identifiers

Variable means its value can vary. You can store any piece of information in a variable. Variables are nothing but just parts of your computer's memory where information is stored. To be identified easily, each variable is given an appropriate name.

Identifiers are names given to identify something. This something can be a variable, function, class, module or other object. For naming any identifier, there are some basic rules like:

- The **first character** of an identifier must be an underscore ('_') or a letter (upper or lowercase).
- The rest of the identifier name can be underscores ('_'), letters (**upper or lowercase**), or digits (**0-9**).
- Identifier names are **case-sensitive**. For example, myvar and myVar are not the same.
- **Punctuation characters** such as @, $, and % are not allowed within identifiers.

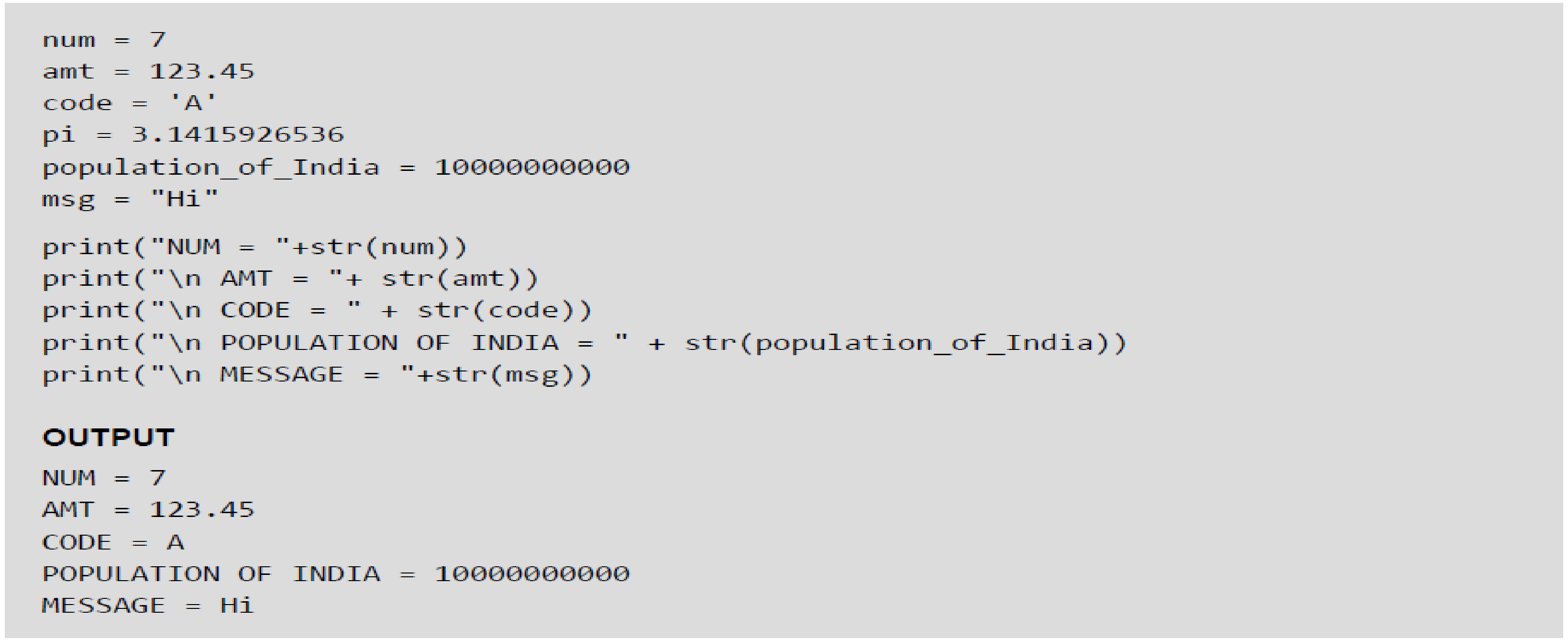
Examples of valid identifier names are sum, __my_var, num1, r, var_20, First, etc.

Examples of invalid identifier names are 1num, my-var, %check, Basic Sal, H#R&A, etc.

Assigning or Initializing Values to Variables

In Python, programmers need not explicitly declare variables to reserve memory space. The declaration is done automatically when a value is assigned to the variable using the equal sign (=). The operand on the left side of equal sign is the name of the variable and the operand on its right side is the value to be stored in that variable.

Example:



```
num = 7
amt = 123.45
code = 'A'
pi = 3.1415926536
population_of_India = 10000000000
msg = "Hi"

print("NUM = "+str(num))
print("\n AMT = "+ str(amt))
print("\n CODE = " + str(code))
print("\n POPULATION OF INDIA = " + str(population_of_India))
print("\n MESSAGE = "+str(msg))
```

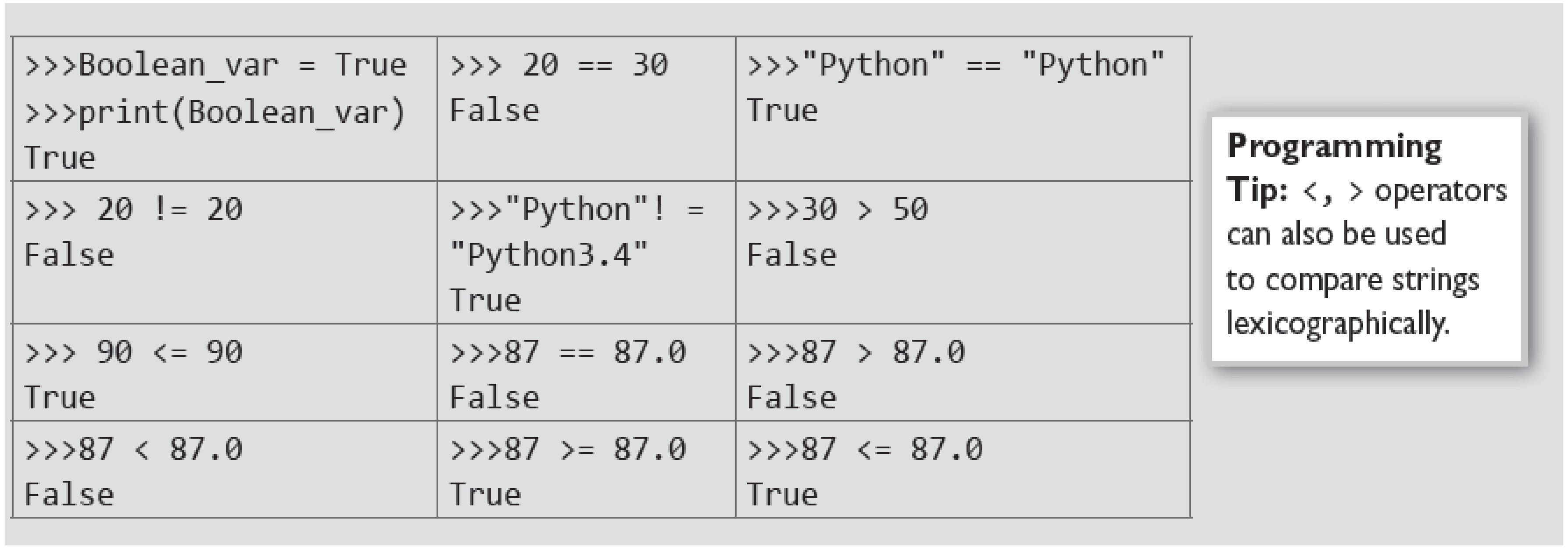
OUTPUT

```
NUM = 7
AMT = 123.45
CODE = A
POPULATION OF INDIA = 10000000000
MESSAGE = Hi
```

Data Type Boolean

Boolean is another data type in Python. A variable of Boolean type can have one of the two values- True or False. Similar to other variables, the Boolean variables are also created while we assign a value to them or when we use a relational operator on them.

Examples:



`>>>Boolean_var = True`
`>>>print(Boolean_var)`
`True`
	`>>> 20 == 30`
	`False`
		`>>>"Python" == "Python"`
		`True`

`>>> 20 != 20`
`False`
	`>>>"Python"! =`
	`"Python3.4"`
	`True`
		`>>>30 > 50`
		`False`

`>>> 90 <= 90`
`True`
	`>>>87 == 87.0`
	`False`
		`>>>87 > 87.0`
		`False`

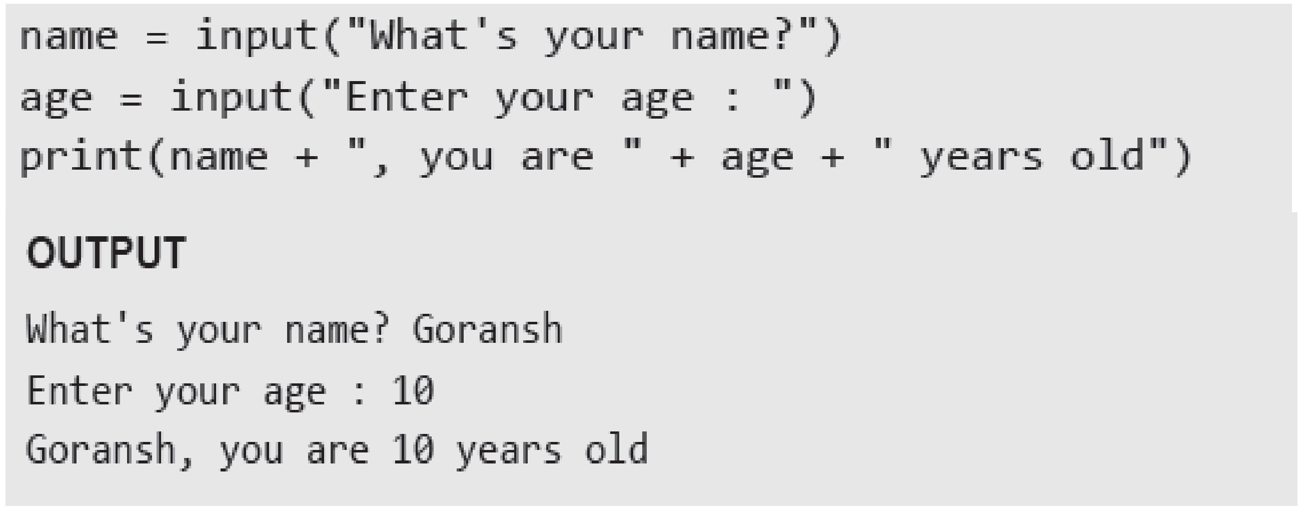
`>>>87 < 87.0`
`False`
	`>>>87 >= 87.0`
	`True`
		`>>>87 <= 87.0`
		`True`

Programming Tip: <, > operators can also be used to compare strings lexicographically.

Input Operation

To take input from the users, Python makes use of the input() function. The input() function prompts the user to provide some information on which the program can work and give the result. However, we must always remember that the input function takes user's input as a string.

Example:



```
name = input("What's your name?")
age = input("Enter your age : ")
print(name + ", you are " + age + " years old")
```

OUTPUT

```
What's your name? Goransh
Enter your age : 10
Goransh, you are 10 years old
```

Comments

Comments are the non-executable statements in a program. They are just added to describe the statements in the program code. Comments make the program easily readable and understandable by the programmer as well as other users who are seeing the code. The interpreter simply ignores the comments.

In Python, a hash sign (#) that is not inside a string literal begins a comment. All characters following the # and up to the end of the line are part of the comment

Example:

```
# This is a comment
print("Hello")  # to display hello
# Program ends here
```

OUTPUT

```
Hello
```

Indentation

Whitespace at the beginning of the line is called *indentation*. These *whitespaces or the indentation* are very important in Python. In a Python program, the leading whitespace including spaces and tabs at the beginning of the logical line determines the indentation level of that logical line.

Example:

```
age = 21
    print("You can vote") # Error! Tab at the start of the line
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2
    print("You can vote")
    ^
IndentationError: unexpected indent
```

Arithmetic Operators

Operator	Description	Example	Output
`+`	Addition: Adds the operands	`>>> print(a + b)`	`300`
`-`	Subtraction: Subtracts operand on the right from the operand on the left of the operator	`>>> print(a – b)`	`-100`
`*`	Multiplication: Multiplies the operands	`>>> print(a * b)`	`20000`
`/`	Division: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient.	`>>> print(b / a)`	`2.0`
`%`	Modulus: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder.	`>>> print(b % a)`	`0`

`//`	Floor Division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (i.e.,rounded away from zero towards negative infinity).	`>>> print(12//5)`
		`>>> print( 12.0//5.0)`
		`>>> print(-19//5)`
		`>>> print(-20.0//3)`
			`2`
			`2.0`
			`-4`
			`-7.0`

`**`	Exponent: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator.	`>>> print(a**b)`	100^{200}

Comparison Operators

Operator	Description	Example	Output
`==`	Returns True if the two values are exactly equal.	`>>> print(a == b)`	`False`
`!=`	Returns True if the two values are not equal.	`>>> print(a != b)`	`True`
`>`	Returns True if the value at the operand on the left side of the operator is greater than the value on its right side.	`>>> print(a > b)`	`False`
`<`	Returns True if the value at the operand on the right side of the operator is greater than the value on its left side.	`>>> print(a < b)`	`True`
`>=`	Returns True if the value at the operand on the left side of the operator is either greater than or equal to the value on its right side.	`>>> print(a >= b)`	`False`
`<=`	Returns True if the value at the operand on the right side of the operator is either greater than or equal to the value on its left side.	`>>> print(a <= b)`	`True`

Unary Operators

Unary operators act on single operands. Python supports unary minus operator. Unary minus operator is strikingly different from the arithmetic operator that operates on two operands and subtracts the second operand from the first operand. When an operand is preceded by a minus sign, the unary operator negates its value.

For example, if a number is positive, it becomes negative when preceded with a unary minus operator. Similarly, if the number is negative, it becomes positive after applying the unary minus operator. Consider the given example.

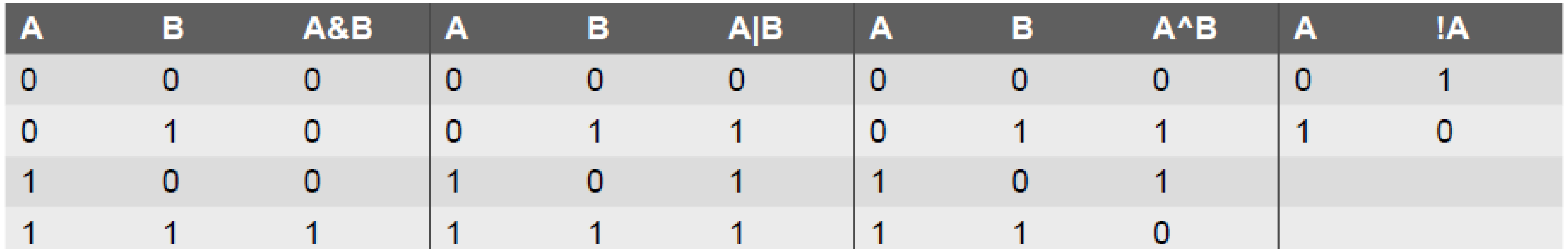
b = 10 a = -(b)

The result of this expression, is a = -10, because variable b has a positive value. After applying unary minus operator (-) on the operand b, the value becomes -10, which indicates it as a negative value.

Bitwise Operators

As the name suggests, bitwise operators perform operations at the bit level. These operators include bitwise AND, bitwise OR, bitwise XOR, and shift operators. Bitwise operators expect their operands to be of integers and treat them as a sequence of bits.

The truth tables of these bitwise operators are given below.



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

Shift Operators

Python supports two bitwise shift operators. They are shift left (<<) and shift right (>>). These operations are used to shift bits to the left or to the right. The syntax for a shift operation can be given as follows:

Examples:

```
operand op num
```

```
if we have x = 0001 1101, then
x << 1 gives result = 0011 1010
```

```
if we have x = 0001 1101, then
x << 4 gives result = 1010 0000
```

```
if we have x = 0001 1101, then
x >> 1 gives result = 0000 1110.
Similarly, if we have x = 0001 1101 then
x << 4 gives result = 0000 0001
```

Logical Operators

Logical AND (&&) operator is used to simultaneously evaluate two conditions or expressions with relational operators. If expressions on both the sides (left and right side) of the logical operator are true, then the whole expression is true. For example, If we have an expression (a>b) && (b>c), then the whole expression is true only if both expressions are true. That is, if b is greater than a and c.

Logical OR (||) operator is used to simultaneously evaluate two conditions or expressions with relational operators. If one or both the expressions of the logical operator is true, then the whole expression is true. For example, If we have an expression (a>b) || (b>c), then the whole expression is true if either b is greater than a or b is greater than c.

Logical not (!) operator takes a single expression and negates the value of the expression. Logical NOT produces a zero if the expression evaluates to a non-zero value and produces a 1 if the expression produces a zero. In other words, it just reverses the value of the expression. For example, a = 10, b b = !a; Now, the value of b = 0. The value of a is not zero, therefore, !a = 0. The value of !a is assigned to b, hence, the result.

Membership and Identity Operators

Python supports two types of membership operators–in and not in. These operators, test for membership in a sequence such as strings, lists, or tuples.

in Operator: The operator returns true if a variable is found in the specified sequence and false otherwise. For example, a in nums returns 1, if a is a member of nums.

not in Operator: The operator returns true if a variable is not found in the specified sequence and false otherwise. For example, a not in nums returns 1, if a is not a member of nums.

Identity Operators

is Operator: Returns true if operands or values on both sides of the operator point to the same object and false otherwise. For example, if a is b returns 1, if id(a) is same as id(b).

is not Operator: Returns true if operands or values on both sides of the operator does not point to the same object and false otherwise. For example, if a is not b returns 1, if id(a) is not same as id(b).

Expressions

An expression is any legal combination of symbols (like variables, constants and operators) that represents a value. In Python, an expression must have at least one operand (variable or constant) and can have one or more operators. On evaluating an expression, we get a value. *Operand* is the value on which operator is applied.

Constant Expressions: One that involves only constants. Example: 8 + 9 – 2

Integral Expressions: One that produces an integer result after evaluating the expression. Example:

a = 10

- *Floating Point Expressions:* One that produces floating point results. Example: a * b / 2
- *Relational Expressions: One that returns either true or false value*. Example: c = a>b
- *Logical Expressions:* One that combines two or more relational expressions and returns a value as *True* or *False*. Example: a>b && y! = 0

- *Bitwise Expressions:* One that manipulates data at bit level. Example: x = y&z
- *Assignment Expressions:* One that assigns a value to a variable. Example: c = a + b or c = 10

Operations on Strings

Examples:

```
>>> print("Missile Man of India" + " - Sir APJ Abdul Kalam")
Missile Man of India - Sir APJ Abdul Kalam
```

```
>>> print("Hello " * 5)
Hello Hello Hello Hello Hello
```

```
str = 'Hello '
print(str + '4')
print(str * 5)
```

OUTPUT

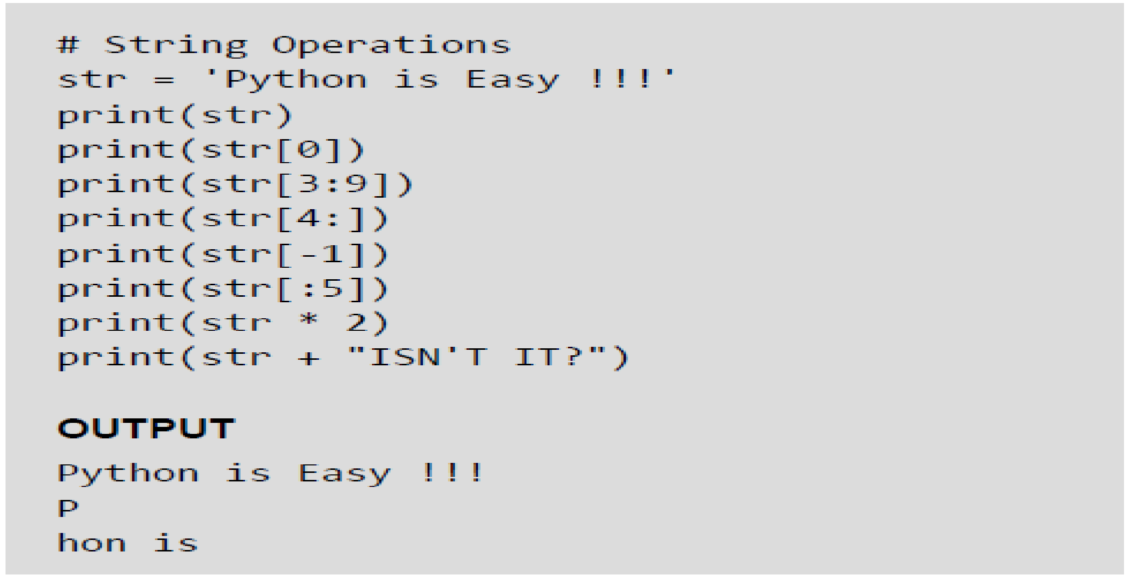
```
Hello 4
Hello Hello Hello Hello Hello
```

Slice Operations on Strings

You can extract subsets of strings by using the slice operator ([ ] and [:]). You need to specify index or the range of index of characters to be extracted. The index of the first character is 0 and the index of the last character is n-1, where n is the number of characters in the string.

If you want to extract characters starting from the end of the string, then you must specify the index as a negative number. For example, the index of the last character is -1.

Examples:



```
# String Operations
str = 'Python is Easy !!!'
print(str)
print(str[0])
print(str[3:9])
print(str[4:])
print(str[-1])
print(str[:5])
print(str * 2)
print(str + "ISN'T IT?")
```

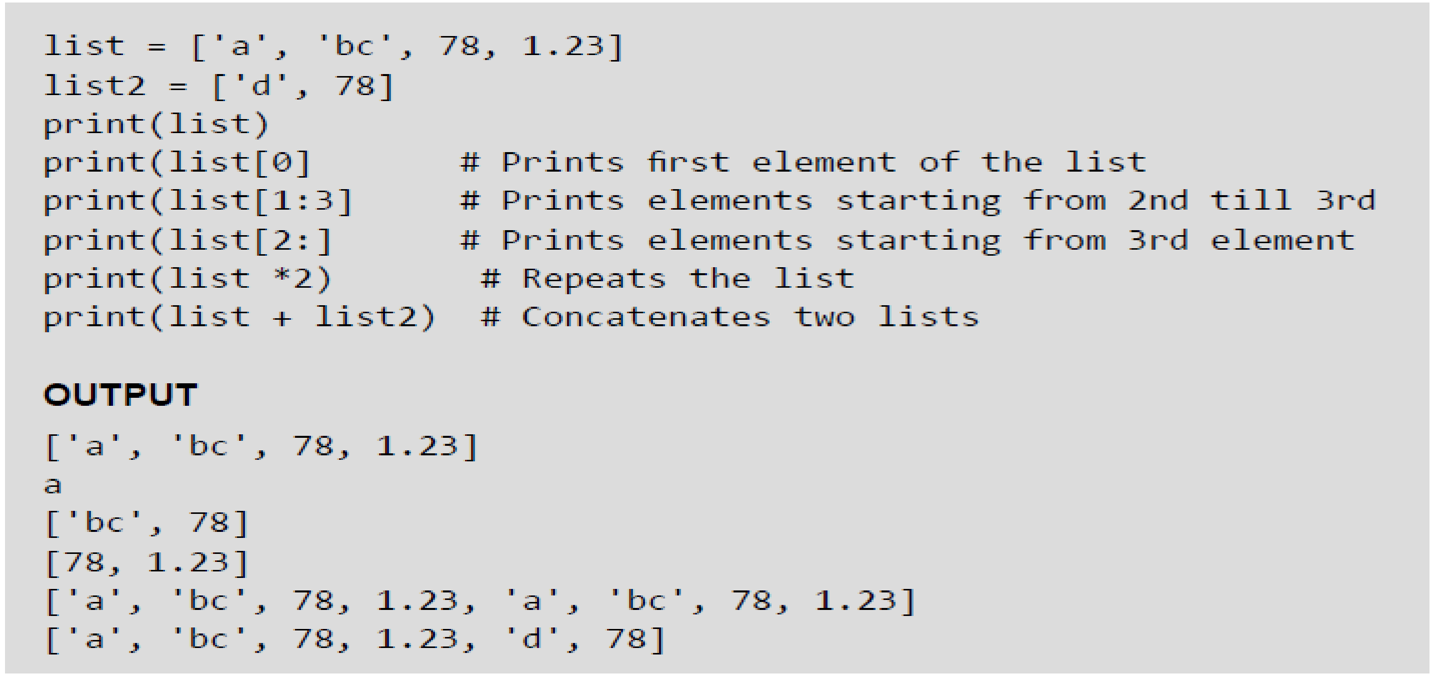
OUTPUT

```
Python is Easy !!!
P
hon is
```

Lists

Lists are the most versatile data type of Python language. A list consist of items separated by commas and enclosed within square brackets The values stored in a list are accessed using indexes. The index of the first element being 0 and n-1 as that of the last element, where n is the total number of elements in the list. Like strings, you can also use the slice, concatenation and repetition operations on lists.

Examples:



```
list = ['a', 'bc', 78, 1.23]
list2 = ['d', 78]
print(list)
print(list[0]       # Prints first element of the list
print(list[1:3]     # Prints elements starting from 2nd till 3rd
print(list[2:]      # Prints elements starting from 3rd element
print(list *2)       # Repeats the list
print(list + list2)  # Concatenates two lists
```

OUTPUT

```
['a', 'bc', 78, 1.23]
a
['bc', 78]
[78, 1.23]
['a', 'bc', 78, 1.23, 'a', 'bc', 78, 1.23]
['a', 'bc', 78, 1.23, 'd', 78]
```

Tuples

A tuple is similar to the list as it also consists of a number of values separated by commas and enclosed within parentheses. The main difference between lists and tuples is that you can change the values in a list but not in a tuple. This means that while tuple is a read only data type, the list is not.

Examples:

```
Tup = ('a', 'bc', 78, 1.23)
Tup2 = ('d', 78)
print(Tup)
print(Tup[0])       # Prints first element of the Tuple
print(Tup[1:3])     # Prints elements starting from 2nd till 3rd
print(Tup[2:])      # Prints elements starting from 3rd element
print(Tup *2)       # Repeats the Tuple
print(Tup + Tup2)  # Concatenates two Tuples
```

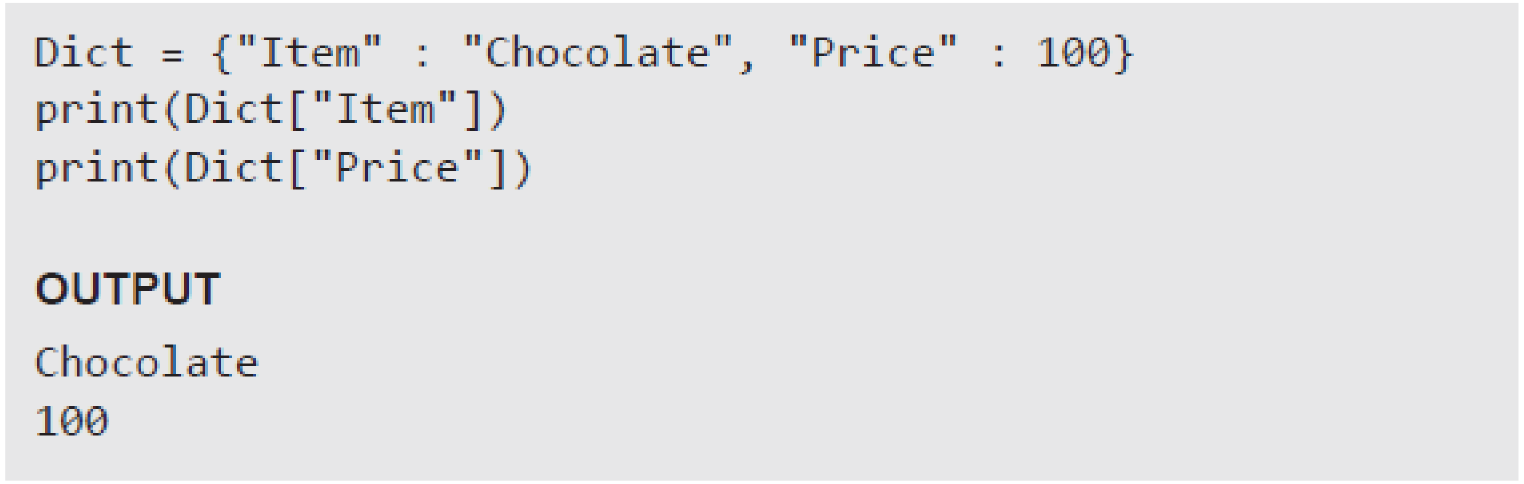
OUTPUT

```
('a', 'bc', 78, 1.23)
a
('bc', 78)
(78, 1.23)
('a', 'bc', 78, 1.23, 'a', 'bc', 78, 1.23)
('a', 'bc', 78, 1.23, 'd', 78)
```

Dictionary

Python's dictionaries stores data in key-value pairs. The key values are usually strings and value can be of any data type. The key value pairs are enclosed with curly braces ({ }). Each key value pair separated from the other using a colon (:). To access any value in the dictionary, you just need to specify its key in square braces ([]).Basically dictionaries are used for fast retrieval of data

Example:



```
Dict = {"Item" : "Chocolate", "Price" : 100}
print(Dict["Item"])
print(Dict["Price"])
```

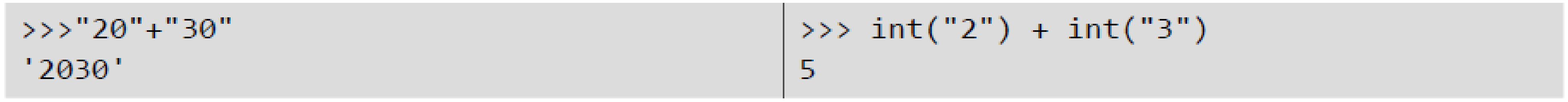
OUTPUT

```
Chocolate
100
```

Type Conversion

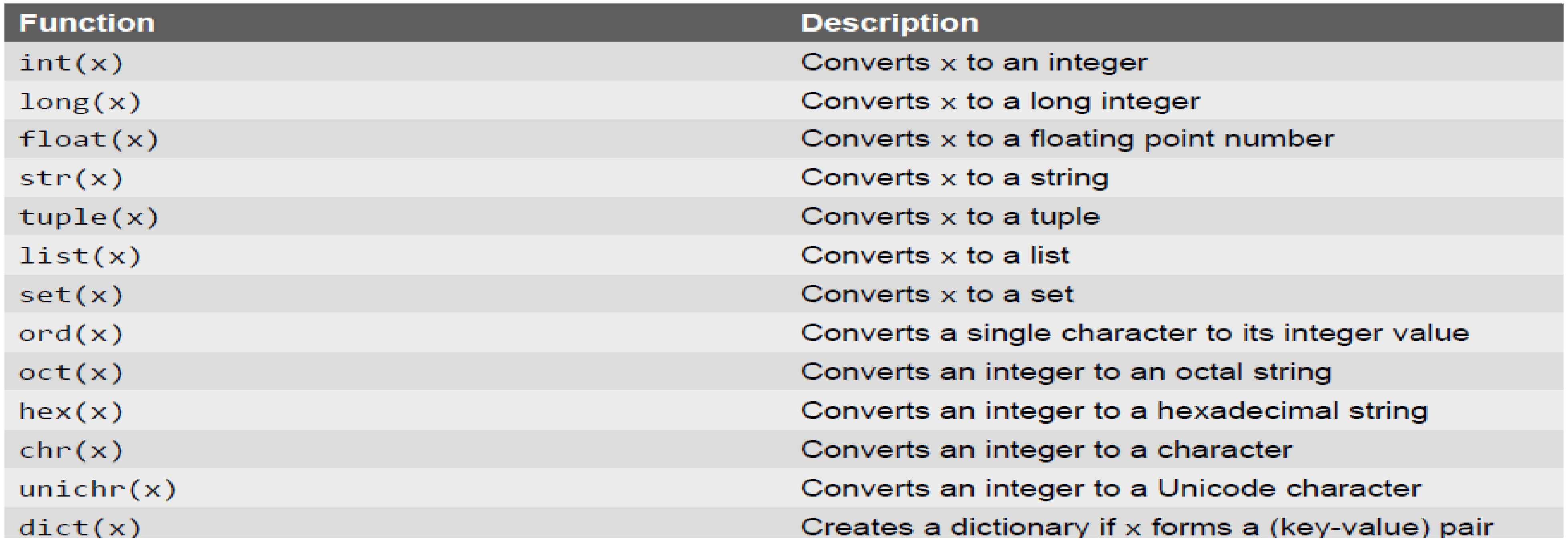
In Python, it is just not possible to complete certain operations that involves different types of data. For example, it is not possible to perform "2" + 4 since one operand is an integer and the other is of string type.

Example:



```
>>>"20"+"30"
'2030'
```

```
>>> int("2") + int("3")
5
```



Function	Description
int(x)	Converts x to an integer
long(x)	Converts x to a long integer
float(x)	Converts x to a floating point number
str(x)	Converts x to a string
tuple(x)	Converts x to a tuple
list(x)	Converts x to a list
set(x)	Converts x to a set
ord(x)	Converts a single character to its integer value
oct(x)	Converts an integer to an octal string
hex(x)	Converts an integer to a hexadecimal string
chr(x)	Converts an integer to a character
unichr(x)	Converts an integer to a Unicode character
dict(x)	Creates a dictionary if x forms a (key-value) pair

Type Casting vs Type Coercion

In the last slide, we have done explicit conversion of a value from one data type to another. This is known as ***type casting***.

However, in most of the programming languages including Python, there is an implicit conversion of data types either during compilation or during run-time. This is also known ***type coercion***. For example, in an expression that has integer and floating point numbers (like 21 + 2.1 gives 23.1), the compiler will automatically convert the integer into floating point number so that fractional part is not lost.