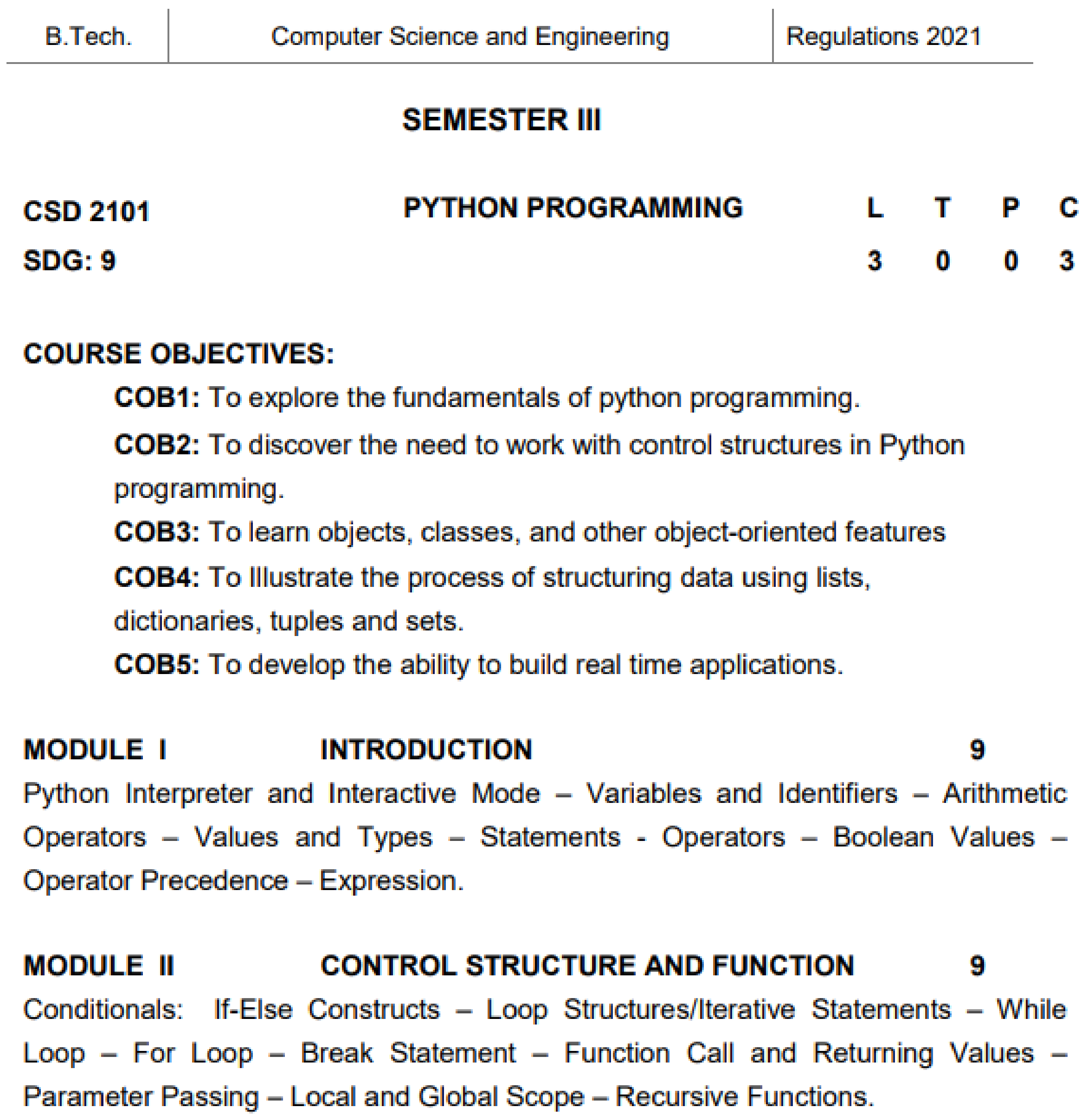
CSD 2101 – Python Programming



B.Tech.	Computer Science and Engineering	Regulations 2021

SEMESTER III

CSD 2101	**PYTHON PROGRAMMING**	**L**	**T**	**P**	**C**
SDG: 9		**3**	**0**	**0**	**3**

COURSE OBJECTIVES:

COB1: To explore the fundamentals of python programming.

COB2: To discover the need to work with control structures in Python programming.

COB3: To learn objects, classes, and other object-oriented features

COB4: To Illustrate the process of structuring data using lists, dictionaries, tuples and sets.

COB5: To develop the ability to build real time applications.

MODULE I INTRODUCTION 9

Python Interpreter and Interactive Mode – Variables and Identifiers – Arithmetic Operators – Values and Types – Statements - Operators – Boolean Values – Operator Precedence – Expression.

MODULE II CONTROL STRUCTURE AND FUNCTION 9

Conditionals: If-Else Constructs – Loop Structures/Iterative Statements – While Loop – For Loop – Break Statement – Function Call and Returning Values – Parameter Passing – Local and Global Scope – Recursive Functions.

CSD 2101 – Python Programming

MODULE III PYTHON DATA STRUCTURE 9

List – Finding and Updating an Item – Nested Lists – Cloning Lists – Looping Through a List – Sorting a List – List Concatenation – List Slices – List Methods – Mutability – Tuples: Creation, Accessing, Updating, Deleting Elements in a Tuple, Tuple Assignment, Tuple as Return Value, Nested Tuples, Basic Tuple Operations – Sets --Dictionary: Creating, Accessing, Adding Items, Modifying, Deleting, Sorting, Looping, Nested Dictionaries Built-in Dictionary Function – Finding Key and Value in a Dictionary.

MODULE IV CLASS AND OBJECTS 9

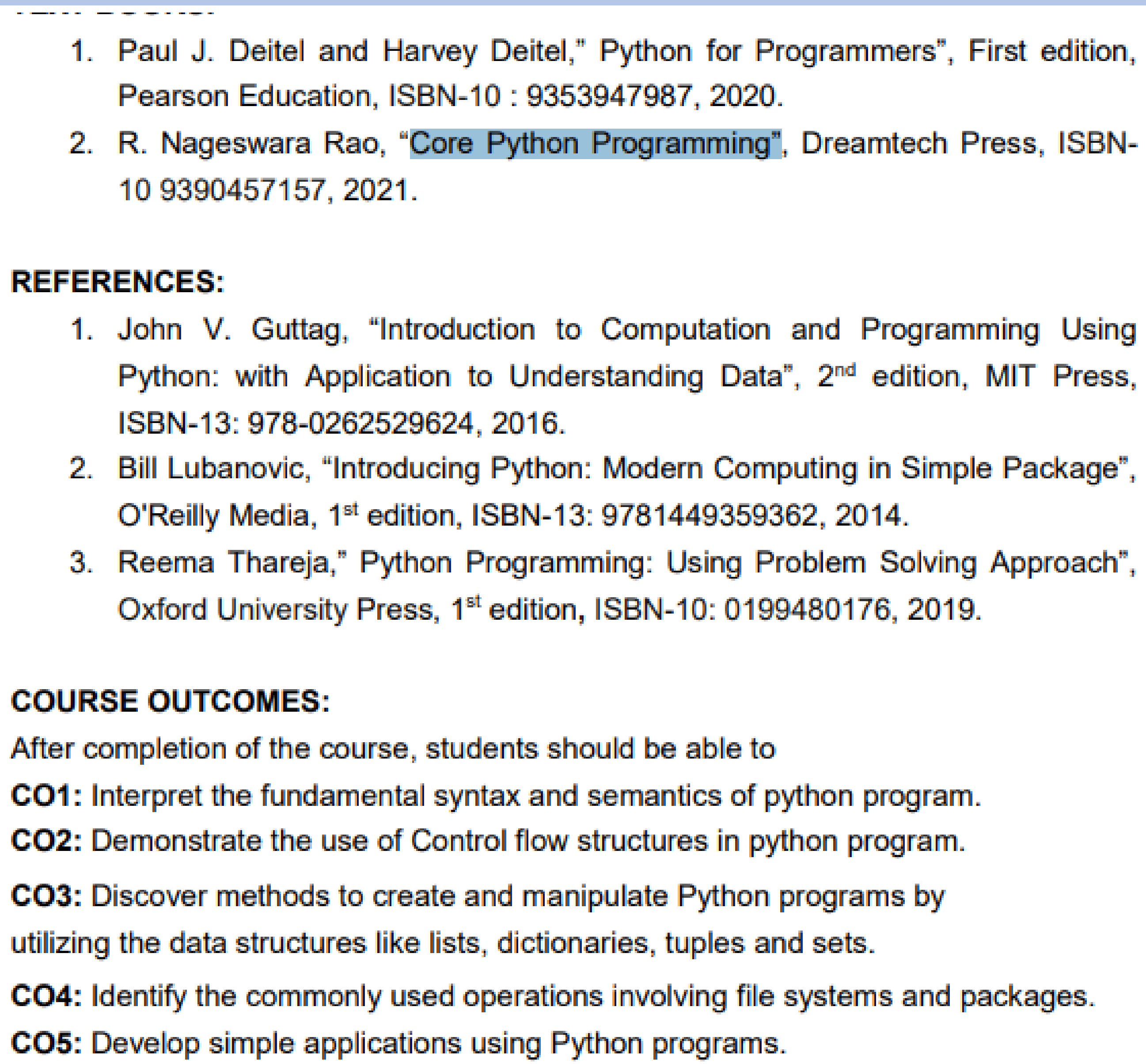
Abstract Data Types – Classes – Inheritance – Multiple level of Inheritance – Substitution Principles – Encapsulation and Information Hiding- Python Standard Libraries–Packages

MODULE V FILE HANDLING AND EXCEPTION HANDLING 9

Introduction to Files – Opening and Closing Files – Reading and Writing Files – File Position – Exception: Errors and Exceptions, Exception Handling, Multiple Exceptions.

L –45; TOTAL HOURS – 45

CSD 2101 – Python Programming



1. Paul J. Deitel and Harvey Deitel," Python for Programmers", First edition, Pearson Education, ISBN-10 : 9353947987, 2020.
2. R. Nageswara Rao, "Core Python Programming", Dreamtech Press, ISBN-10 9390457157, 2021.

REFERENCES:

1. John V. Guttag, "Introduction to Computation and Programming Using Python: with Application to Understanding Data", 2nd edition, MIT Press, ISBN-13: 978-0262529624, 2016.
2. Bill Lubanovic, "Introducing Python: Modern Computing in Simple Package", O'Reilly Media, 1st edition, ISBN-13: 9781449359362, 2014.
3. Reema Thareja," Python Programming: Using Problem Solving Approach", Oxford University Press, 1st edition, ISBN-10: 0199480176, 2019.

COURSE OUTCOMES:

After completion of the course, students should be able to

CO1: Interpret the fundamental syntax and semantics of python program.

CO2: Demonstrate the use of Control flow structures in python program.

CO3: Discover methods to create and manipulate Python programs by utilizing the data structures like lists, dictionaries, tuples and sets.

CO4: Identify the commonly used operations involving file systems and packages.

CO5: Develop simple applications using Python programs.

Operators

- Arithmetic Operators
- Comparison Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Identity Operators
- Membership Operators

Arithmetic Operators a = 5 b=3

+	Addition	8
-	Subtraction	2
*	Multiplication	15
/	Division	1.6
%	Modulo	2
**	Exponent	125
//	Floor Division	1

Comparison Operators   a = 5   b=3

==	Equal to	False
!=	Not Equal	True
>	Greater than	True
<	Less than	False
>=	Greater than or equal	True
<=	Less than or equal	False

Logical Operators a = 3

and	a<5 and a<10	True
or	a>2 or a<2	True
not	not a	False

Assignment Operators a = 5 b=3

+=	a+=b	8
-=	a-=b	2
*	a*=b	15
/	a/=b	1.6
%	a%= b	2
**	a**=b	125
//	a//=b	1

Bitwise Operators a = 3 b=5

&	AND	if(a & b)	False
|	OR	if(a | b)	True
^	XOR	a^b	0011^0101=0110
~	NOT	~a	-3 + (-1) = -4
<<	Zero fill left shift		0011 (a<<2) = 001100
>>	signed right shift		0011 (a>>1) = 001

Identity Operators

is – True is the operands are identical

is not – True is the operands are not identical

Both are used to check if two values are located on the same part of the memory.

a = 3 b = 2 c = a

Print (a is c) True

Print (a is not c) False

Print (a is not b) True

Membership Operators

in – True if value is found in the sequence

Not in – True if value is not found in the sequence

used to whether a value or variable is in a sequence.

a = 5

List = [1,2,3,4,5]

if(a in list) True

if(a not in list) False

Statements

– **are code instruction that are executed by the Python interpreter.**

- Simple

- Multiple

- Complex

Simple:

- Python simple statement is comprised of a single line.

1. Python Expression Statement

```
a = int('1')
Sum = a + b
```

2. Python Assignment Statement

```
a = 10
Text = “hello”
```

3. Python Assert Statement (try-except-clause)

```
assert 5<10
assert (True or False)
```

4. Python pass Statement

```
def sum():
    pass
```

5. Python del Statement

```
string = “hello”
del string
```

6. Python return Statement

```
def sum():
        return 10
```

7. Python yield Statement

```
def sum():
yield ‘statement1’
```

- Used to create a generator function
- can be used only inside function body
- It automatically becomes a generator function.

8. Python raise Statement

```
def sum():
        raise TypeError(“Exception Error”)

def divide(x,y):
   print(f'{x}/{y} is {x/y}')

divide(10,2)
10/2 is 5.0
divide(10,0)
```

9. Python break Statement

```
num=[1,2,3]
for n in num:
        if n>2:
 break
```

10. Python continues Statement

```
num=[1,2,3]
for n in num:
   if n>2:
      continue
   print(n)

1
2
```

11. Python import Statement

```
import collections
import calendar as cal
```

12. Python global Statement

```
name = "mohamed"
def global_name():
    global name
    name = "yousuff"

print(name)
mohamed
global_name()
print(name)
yousuff
```

Python Multi-line Statements

Python statements are usually written in a single line. The newline character marks the end of the statement. If the statement is very long, we can explicitly divide it into multiple lines with the line continuation character (\).

```
Text = “hello”\
          “welcome”\
          “second CSE”
Sum = 1+2\
          3+4
```

Python Compound Statements

Python compound statements contain a group of other statements and affect their execution. The compound statement generally spans multiple lines.

1. Python if Statement

```
if 5<10:
    print(“True”)
else:
    print(“False)
```

2. Python for Statement

```
for n in (1,2,3,4,5):
          print(n)
```

3. Python while Statement

```
        count=0
while count<3:
    print(count)
    count+=1
    0
    1
    2
```

4. Python try Statement

```
try:
    print("try")
except ValueError as ve:
    print(ve)
```

```
try
```

5. Python with Statement

```
with open('data.csv') as file:
        file.read()
```

6. Python Function Definition Statement

```
def sum():
    print(‘hello’)
```

7. Python Class Definition Statement

```
class Data:
    id = 0
```

CSD 2101 – Python Programming

Boolean Values

Python bool()

- takes a specified argument and returns its boolean value.

```
bool(None)
False

year="second"
print(year, 'is', bool(year))
second is True
```

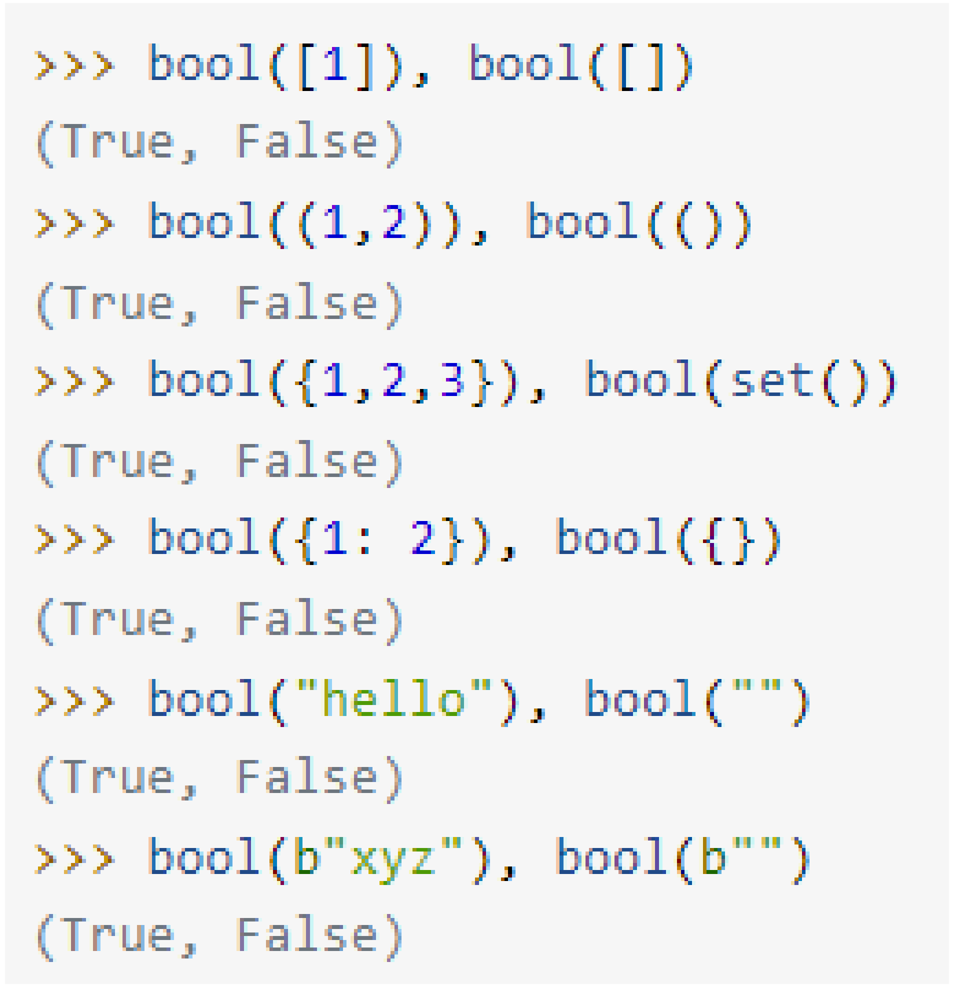
Numbers as Boolean Values

```
bool(5),bool(0),bool(-3)
(True, False, True)
```

CSD 2101 – Python Programming

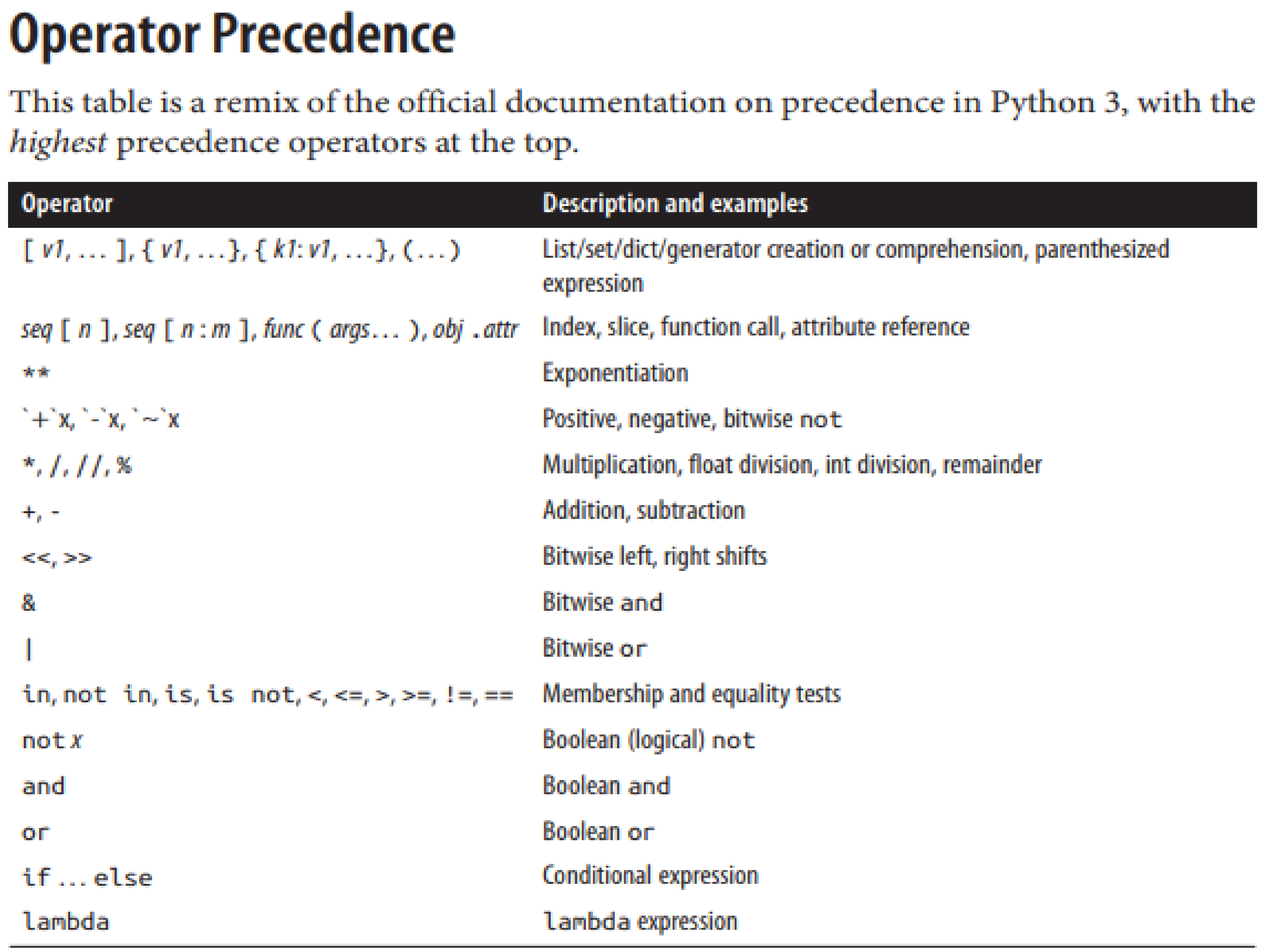
Sequences as Boolean Values

```
bool(0.2+0.5+(-5.0))
True
bool(fractions.Fraction("1/2"))
True
bool(fractions.Fraction("2/1"))
True
bool(fractions.Fraction("0/2"))
False
```



```
>>> bool([1]), bool([])
(True, False)
>>> bool((1,2)), bool(())
(True, False)
>>> bool({1,2,3}), bool(set())
(True, False)
>>> bool({1: 2}), bool({})
(True, False)
>>> bool("hello"), bool("")
(True, False)
>>> bool(b"xyz"), bool(b"")
(True, False)
```

CSD 2101 – Python Programming



Operator Precedence

This table is a remix of the official documentation on precedence in Python 3, with the *highest* precedence operators at the top.

Operator	Description and examples
[*v1*, …], { *v1*, …}, { *k1*: *v1*, …}, (…)	List/set/dict/generator creation or comprehension, parenthesized expression
seq [*n*], *seq* [*n* : *m*], *func* (*args*…), *obj* .*attr*	Index, slice, function call, attribute reference
**	Exponentiation
`+`x, `-`x, `~`x	Positive, negative, bitwise not
*, /, //, %	Multiplication, float division, int division, remainder
+, -	Addition, subtraction
<<, >>	Bitwise left, right shifts
&	Bitwise and
|	Bitwise or
in, not in, is, is not, <, <=, >, >=, !=, ==	Membership and equality tests
not *x*	Boolean (logical) not
and	Boolean and
or	Boolean or
if … else	Conditional expression
lambda	lambda expression

Thank You