

Computational Thinking with Programming



Lecture - 4

Operators, Expression and Data types

Today's Outline

- Previous Session:

- Introduction to Token.
- Statements and Expressions.

- Today's Session:

- Data Types and Operators
 - Mutable and Immutable data types.
 - Data type conversion.
 - Operators.

- Hands on Session with Jupyter Notebook:

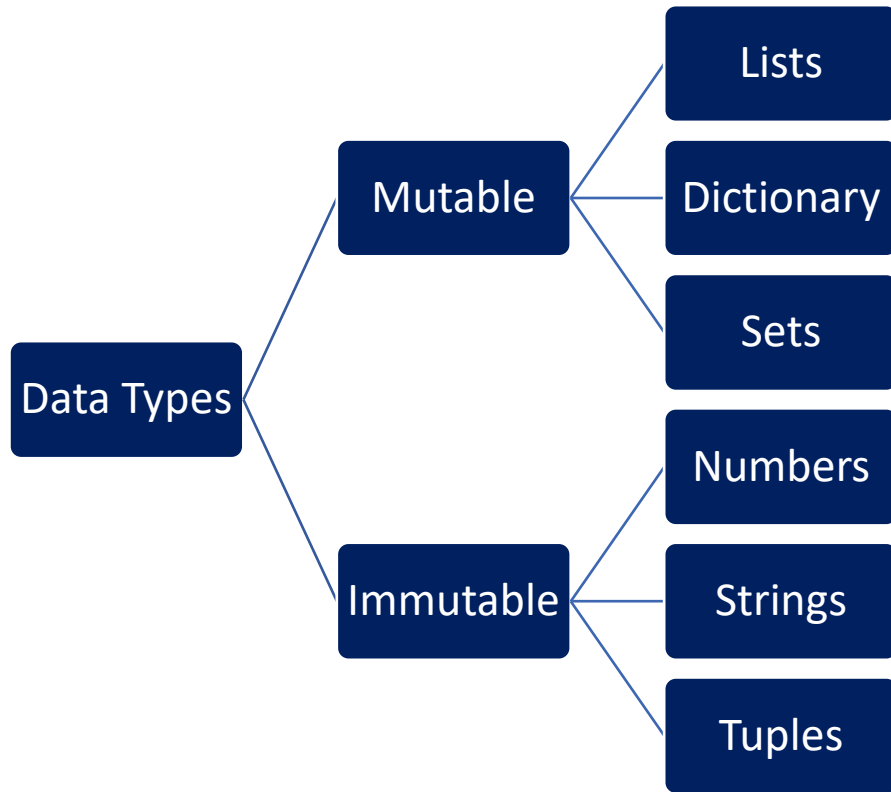
- We will practice on the Python basic elements in Jupyter Notebook.

Data Types

- Data type is the classification of the type of values that can be assigned to variables.
- Dynamically typed languages, the interpreter itself predicts the data type of the Python Variable based on the type of value assigned to that variable.
 - x is a variable and 2 is its value
 - x can be assigned different values; hence, its type changes accordingly

```
>>> x = 2
>>> type(x)
int
>>> x = 2.3
>>> type(x)
float
```

Data Types



- **Mutable Data Types:** Data types in python where the value assigned to a variable can be changed
- **Immutable Data Types:** Data types in python where the value assigned to a variable cannot be changed

Data Types

- **Numbers:** The number data type in Python is used to store numerical values. It is used to carry out the normal mathematical operations.
- **Strings:** Strings in Python are used to store textual information. They are used to carry out operations that perform positional ordering among items.
- **Lists:** The list data type is the most generic Python data type. Lists can consist of a collection of mixed data types, stored by relative positions.
- **Tuples:** Tuples are one among the immutable Python data types that can store values of mixed data types. They are basically a list that cannot be changed.
- **Sets:** Sets in Python are a data type that can be considered as an unordered collection of data without any duplicate items.
- **Dictionaries:** Dictionaries in Python can store multiple objects, but unlike lists, in dictionaries, the objects are stored by keys and not by positions.

Data Types (Numbers)

The number data type is divided into the following five data types:

- Integer
- Long Integer (removed from py3)
- Octal and Hexadecimal
- Floating-point Numbers
- Complex Numbers

```
>>> a = 2
>>> type(a)
int
>>> a = 2.5
>>> type(a)
float
>>> a = 0o11
>>> type(a)
int
```

```
>>> a = 0x19
>>> type(a)
int
>>> a = 2 + 5j
>>> type(a)
complex
>>> a = 99999999L
>>> type(a)
long
```

Data Types (Numbers)

- We can use the following built-in functions to convert one number type into another:
- `int(x)`, to convert `x` into an integer value
- `long(x)`, to convert `x` into a long integer value
- `float(x)`, to convert `x` into a floating-point number
- `complex(x)`, to convert `x` into a complex number where the imaginary part stays 0 and `x` becomes the real part
- `complex(x,y)`, to convert `x` and `y` to a complex number where `x` becomes the real part and `y` becomes the imaginary part

Data Types (String)

- Python string is an ordered collection of characters which is used to represent and store the text-based information. Strings are stored as individual characters in a contiguous memory location. It can be accessed from both directions: forward and backward.

12

```
>>> a = "Bennett"  
>>> print(a)  
Bennett  
>>> a = 'University'  
>>> print(a)  
University
```


Data Types (String)

- Characters of string can be individually accessed using a method called indexing.
- Characters can be accessed from both directions: forward and backward.
- Forward indexing starts from 0, 1, 2.... Whereas, backward indexing starts from -1, -2, -3...

```
>>> a = "Bennett University"  
>>> print(a[5])  
t  
>>> print(a[-1])  
y  
>>> print(a[-5])  
r
```

Data Types (Tuples)

- Tuple data type in Python is a collection of various immutable Python objects separated by commas.
- Tuples are generally used for different Python Data Types.
- A Python tuple is created using parentheses around the elements in the tuple. Although using parentheses is only optional, it is considered a good practice to use them.

```
>>> a = (1,2,3,4)
>>> print(a)
(1,2,3,4)
>>> a = ('ABC','DEF','XYZ')
>>> print(a)
(ABC,DEF,XYZ)
```

Data Types (Tuple)

- To access an element of a tuple, we simply use the index of that element. We use square brackets.
- Reverse Indexing by using indexes as -1 , -2 , -3 , and so on, where -1 represents the last element.
- Slicing that is, extract some elements from the tuple.

```
>>> a = (1,2,3,4)
>>> print(a[1])
2
>>> a = ('ABC','DEF','XYZ')
>>> print(a[2])
XYZ
```

```
>>> a = (1,2,3,4)
>>> print(a[-1])
4
>>> a = ('ABC','DEF','XYZ')
>>> print(a[1:])
(DEF, XYZ)
```

Data Types (List)

Unlike strings, lists can contain any sort of objects: numbers, strings, and even other lists. Python lists are:

- Ordered collections of arbitrary objects
- Accessed by offset
- Arrays of object references
- Of variable length, heterogeneous, and arbitrarily nestable
- Of the category, mutable sequence
- Data types in which elements are stored in the index basis with starting index as 0
- Enclosed between square brackets ‘[]’

```
>>> a = [2,3,4,5]
>>> b =
["Bennett",
"University"]
>>> print(a,b)
>>>
[2,3,4,5]['Bennett',
'University']
```

Data Types (List)

- Much similar to strings, we can use the index number to access items in lists as shown below.
- Accessing a List Using Reverse Indexing
 - To access a list in reverse order, we have to use indexing from -1 , -2 Here, -1 represents the last item in the list.

```
>>> a = ["Bennett", "University", "Computer"]
>>> print(a[0])
Bennett
>>> print(a[-1])
Computer
>>> print(a[1])
University
```

Data Types (Set)

- It is an ~~ordered~~ unordered collection of elements which means that a set is a collection that stores elements of different Python Data Types.
- In Python sets, elements don't have a specific order.
- Sets in Python can't have duplicates. Each item is unique.
- The elements of a set in Python are **immutable**. They can't accept changes once added.

```
>>> myset = {"bennett", "computer", "science"}  
>>> print(myset)  
{'bennett', 'computer', 'science'}  
>>> myset = set(("bennett", "computer", "science"))
```

Data Types (Dictionary)

- Yet another unordered collection of elements.
- Difference between dictionary and other unordered Python data types such as sets lies in the fact that unlike sets, a dictionary contains keys and values rather than just elements.
- Unlike lists the values in dictionaries are accessed using keys and not by their positions

```
>>>dict1={"Branch":"computer","College":"Bennett","year":2020}  
>>>print (dict1)  
{'Branch':'computer','College':'Bennett','year':2020}  
>>>di = dict({1:'abc',2:'xyz'})
```

Data Types (Dictionary)

- The keys are separated from their respective values by a colon (:) between them, and each key-value pair is separated using commas (,).
- All items are enclosed in curly braces.
- While the **values in dictionaries may repeat, the keys are always unique.**
- The **value can be of any data type, but the keys should be of immutable data type, that is**
- Using the key inside square brackets like we used to use the index inside square brackets.

```
>>>dict1={"Branch": "computer", "College": "Bennett", "year": 2020}  
>>>print (dict1[year])  
2020
```


Datatype Conversion

- As a matter of fact, we can do various kinds of conversions between strings, integers and floats using the built-in *int*, *float*, and *str* functions

```
>>> x = 10
>>> float(x)
10.0
>>> str(x)
'10'
>>>
```

integer → float
integer → string

```
>>> y = "20"
>>> float(y)
20.0
>>> int(y)
20
>>>
```

string → float
string → integer

```
>>> z = 30.0
>>> int(z)
30
>>> str(z)
'30.0'
>>>
```

float → integer
float → string

Explicit and Implicit Data Type Conversion

- Data conversion can happen in two ways in Python
 1. **Explicit Data Conversion** (we saw this earlier with the *int*, *float*, and *str* built-in functions)
 2. **Implicit Data Conversion**
 - Takes place *automatically* during run time between *ONLY* numeric values
 - E.g., Adding a float and an integer will automatically result in a float value
 - E.g., Adding a string and an integer (or a float) will result in an *error* since string is not numeric
 - Applies type promotion to avoid loss of information
 - Conversion goes from integer to float (e.g., upon adding a float and an integer) and not vice versa so as the fractional part of the float is not lost

Implicit Data Type Conversion: Examples

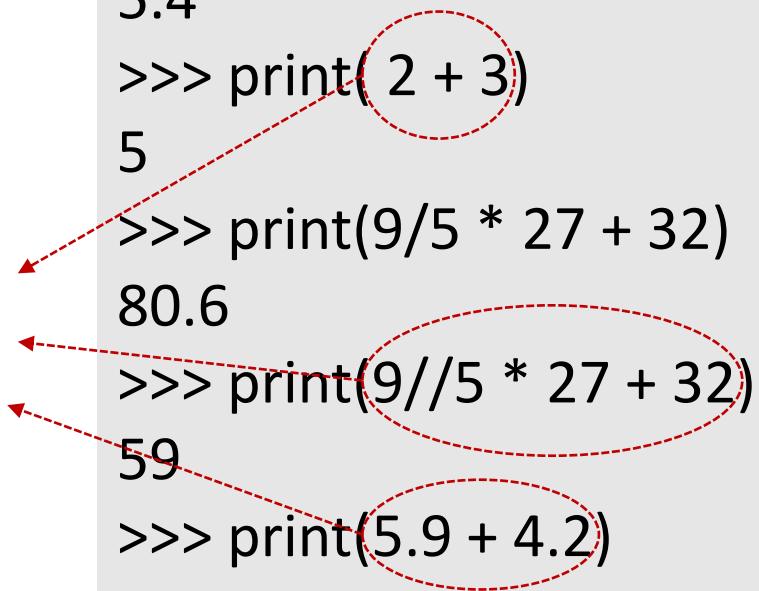
- The result of an expression that involves a float number alongside (an) integer number(s) is a float number

```
>>> print(2 + 3.4)
5.4
>>> print( 2 + 3)
5
>>> print(9/5 * 27 + 32)
80.6
>>> print(9//5 * 27 + 32)
59
>>> print(5.9 + 4.2)
10.100000000000000001
>>>
```

Implicit Data Type Conversion: Examples

- The result of an expression that involves a float number alongside (an) integer number(s) is a float number
- The result of an expression that involves values of the same data type will not result in any conversion

```
>>> print(2 + 3.4)
5.4
>>> print(2 + 3)
5
>>> print(9/5 * 27 + 32)
80.6
>>> print(9//5 * 27 + 32)
59
>>> print(5.9 + 4.2)
10.1000000000000001
>>>
```




Operators

- Arithmetic Operators (`**`, `*`, `/`, `//`, `%`, `+`, `_`)
- Comparison Operators (`<`, `<=`, `>`, `>=`, `==`)
- Python Assignment Operators (`=`, `+=`, `-=`, `*=`, `/=`)
- Logical Operators (`and`, `or`, `not`)
- Bitwise Operators (`&`, `|`, `~`, `>>`, `<<`, `^`)
- Membership Operators (`in`, `not in`)

Operator

- Arithmetic operator

Precedence						
Low		High				
						
<div><div><div>+</div></div></div>	-	*	/	//	%	**
Plus	minus	multiplication	Float division	Integer division	Mod (remainder)	power
3	3	2	2	2	2	1

Left to right

Right to left

Lets solve -

$$5/10*5+5*2$$

Ans: 12.5

$$5//10*5+5*2$$

Ans: 10

525 + 10

$$5\%10*5+5*2$$

Ans: 35

$$2^{**3}*\underline{1}$$

Ans: 8

$$\underline{1}^{**3}*\underline{2}$$

Ans: 1

$$\underline{2}^{**1}*\underline{3}$$

Ans: 2

Comparison Operators

Boolean expressions ask a question

- Produce a Yes or No result which we use to control program flow

Boolean expressions using comparison operators evaluate to:

- True / False - Yes / No

Comparison operators look at variables

- But do not change the variables

Operator	Description
==	Equals to
!=	Not equals to
<>	Not equals to
>	Greater than
<	Less Than
>=	Greater Than Equals to
<=	Less Than Equals to

Example

```
x=5
print(x>3) True
print(x==5) True
print(x>=5) True
print(x<=5) True
print(x<6) True
print(x!=6) True
```

```
a = 5
b = 5
c = 3
print(a = b)
print(a = c)
print(b = c)
```

```
a = 5
b = 3
c = 4
d = 4
e = 5
print(a == b) False
print(a == c) False
print(a == d) False
print(a == e) True
```

TypeError Traceback (most recent call last)
<ipython-input-2-93bdf5d98946> in <module>()

```
2 b = 5
3 c = 3
----> 4 print(a = b)
5 print(a = c)
6 print(b = c)
```

TypeError: 'a' is an invalid keyword argument for this function

Logical operators

- Logical operators are the and, or, not operators.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Logical operators AND/OR

AND		
X	Y	Output
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

```
onList = True
inStock = True
onSale = False
rotten = False
#The lines below
```

```
>>>a=7>7 and 2>-1
>>>print(a)
FALSE
```

```
>>>a=7>7 or 2>-1
>>>print(a)
True
```

```
>>>print(7 and 0 or 5)
5
```

```
print(onList and inStock)
```

```
print(onList and onSale)
```

```
print(onSale and rotten)
```

```
print(onList and inStock and onSale)
```

```
print(onList and inStock and onList)
```

```
print(onList and inStock)
```

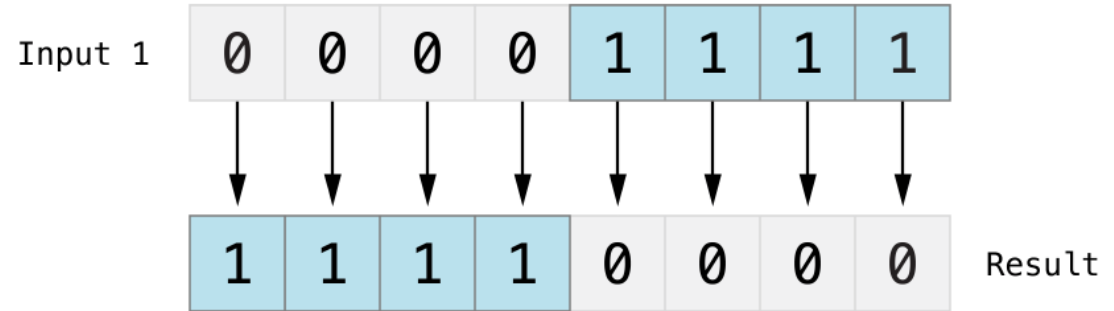
OR		
X	Y	Output
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Logical operators

NOT

Just the reverse of what is there.

- `not(true)` → **false**



```
bool_1 = not (True or True)
bool_2 = True and (False or False)
bool_3 = True or (False or False)
bool_4 = (True or not True) and (True and True)
bool_5 = (3>5) or (5<4 and not 5>=7)
```

```
print(bool_1)
print(bool_2)
print(bool_3)
print(bool_4)
print(bool_5)
```

False
False
True
True
False

Bitwise operators

Bitwise operators act on operands as if they were string of binary digits.

It operates bit by bit.

- 2 is 10 in binary and 7 is 111.

In the table below:

- Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Description
& Binary AND	Operator copies a bit to the result if it exists in both operands
 Binary OR	It copies a bit if it exists in either operand.
^ Binary XOR	It copies the bit if it is set in one operand but not both.
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.

Bitwise operators AND

5 & 7

- Same as 101 & 111.
- This results in 101
- Which is binary for 5.

4 | 8

- Binary for 4 is 0100, and that for 8 is 1000.
- After operation, output is 1100
- Which is binary for 12.

```
print(5&7).
```

5

Operation	Output
0 & 0	0
0 & 1	0
1 & 0	0
1 & 1	1

Operation	Output
0 0	0
0 1	1
1 0	1
1 1	1

```
print(4|8)
```

12

```
"{0:b}".format(5).
```

'101'

Bitwise operators

XOR

XOR (exclusive OR) returns 1

- If one operand is 0 and another is 1.

Otherwise, it returns 0.

Operator	Output
0^0	0
0^1	1
1^0	1
1^1	0

```
print(5^3)
```

6

Shift operator

Left Shift (<<)

```
In [60]: 12<<1
```

```
Out[60]: 24
```

Right Shift (>>)

```
In [61]: 12>>1
```

```
Out[61]: 6
```

```
In [62]: 12<<2
```

```
Out[62]: 48
```

```
In [63]: 12>>2
```

```
Out[63]: 3
```

Let's Solve

15^{13}

$12 \& 10$

$11 | 00$

$5<<2$

$5>>2$

Ans.

2

8

11

20

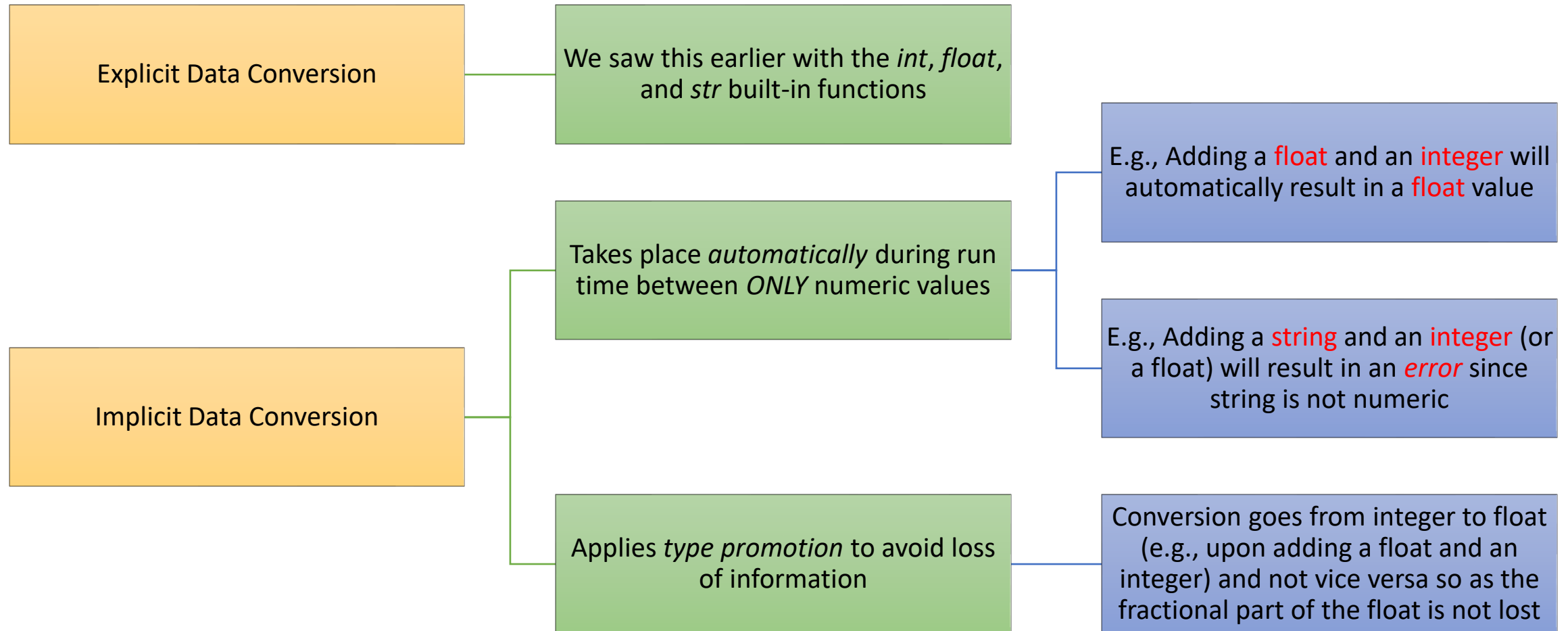
1

Operator Precedence

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
==, !=, >, >=, <, <=, in, not in	Comparisons, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

DATA TYPE CONVERSION

Explicit and Implicit



Let's Solve

Operators
()
**
+x, -x, ~x
*, /, //, %
+, -
<<, >>
&
^
==, !=, >, >=, <, <=, in, not in
not
and
Or
=, *=, /=, //=, %=

$10 * 4 >> 2$ and 3

Ans: 3

```
5|10&12>>2
```

Ans: 7

$10 \%(15 < 10 \text{ and } 20 < 30)$

Ans: Error: integer division or modulo by zero

```
A=5
A+=10//5
print(A)
```

Ans: 7

$10 / (5 - 5)$

Ans: Error: division by zero

```
A=5
A**=10//5
print(A)
```

Ans: 25

$2.5 \% 0.15$

Ans: 0.10000000000000009

```
a=7
a%=6.5/7<=7 and 4<=6
a
```

Ans: 0

Thank You

?