Computing (ES 112)

Yogesh K. Meena
Shouvick Mondal

August 2024

Computer Science & Engineering
IIT Gandhinagar

Lecture 4 (contd.)

Variables, Expressions, and Statements

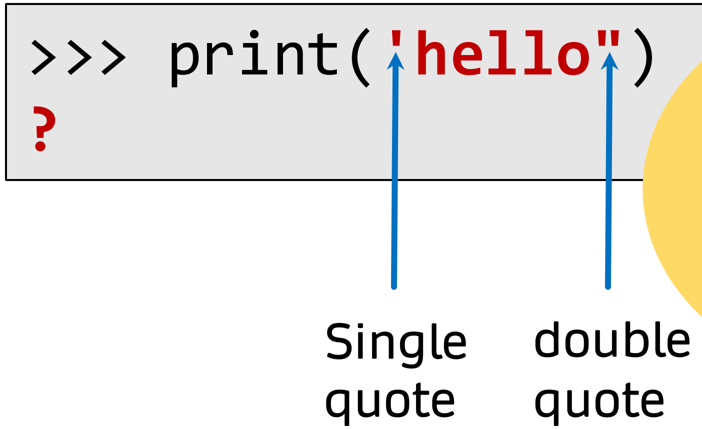
Constants

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Fixed values such as numbers, letters, and strings, are called “constants” because their value does not change

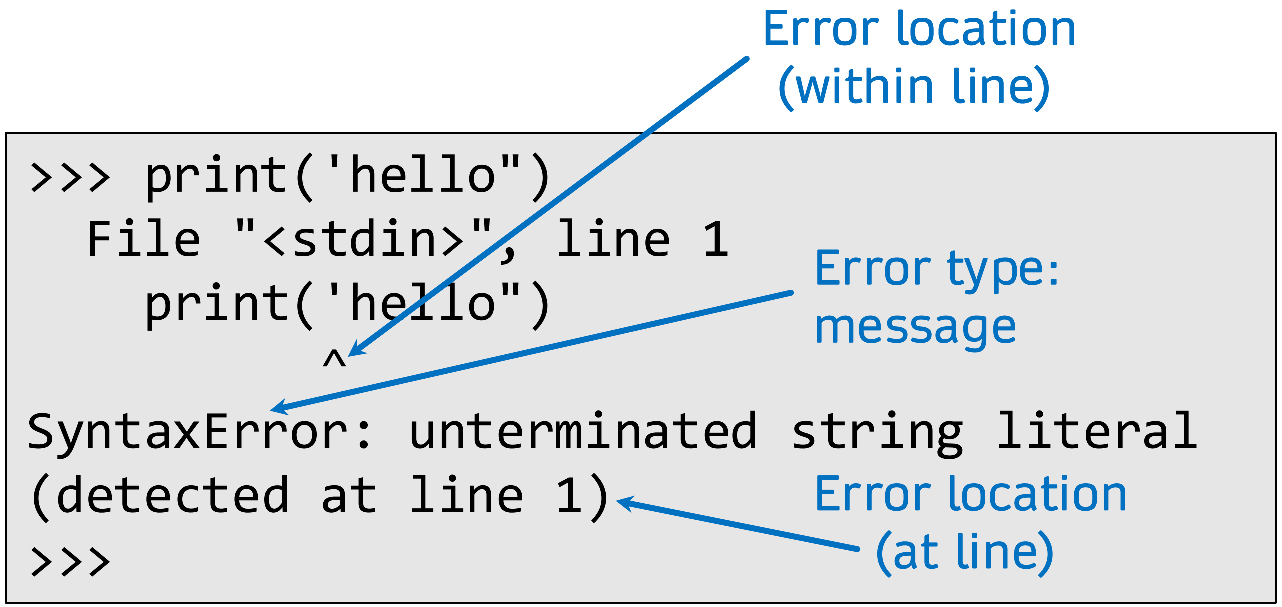
Numeric **constants** are as you expect

String **constants** use single quotes (') or double quotes (")



```
>>> print('hello")
?
```

What will be the output of the program?



```
>>> print('hello")
  File "<stdin>", line 1
    print('hello")
          ^
SyntaxError: unterminated string literal
(detected at line 1)
>>>
```

Reserved Words

You cannot use **reserved words** as variable names / identifiers

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

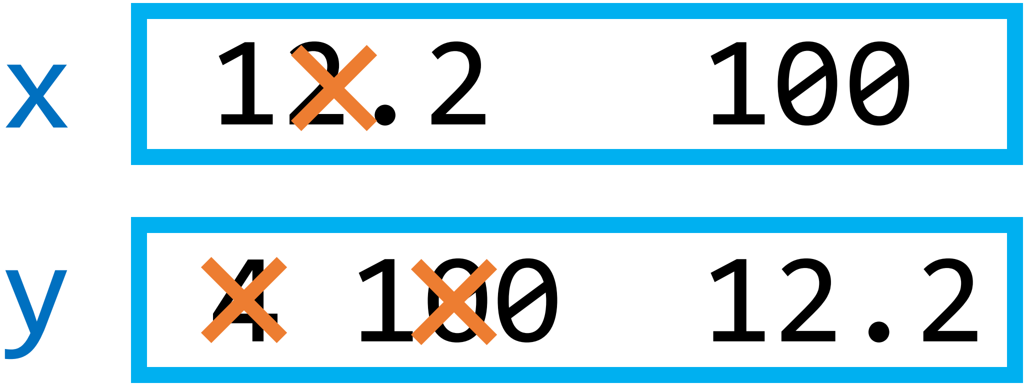
These are predefined special constants in Python

Variables and Assignment Statements

```
x = 12.2
y = 4
y = 100
x,y = y,x
```

- A variable is a "name"d place in the memory where a programmer can store (write) data and later retrieve (read/load) data by just using the "name"
- Programmers get to choose the names of the variables. You can change the contents of a variable in a later statement.

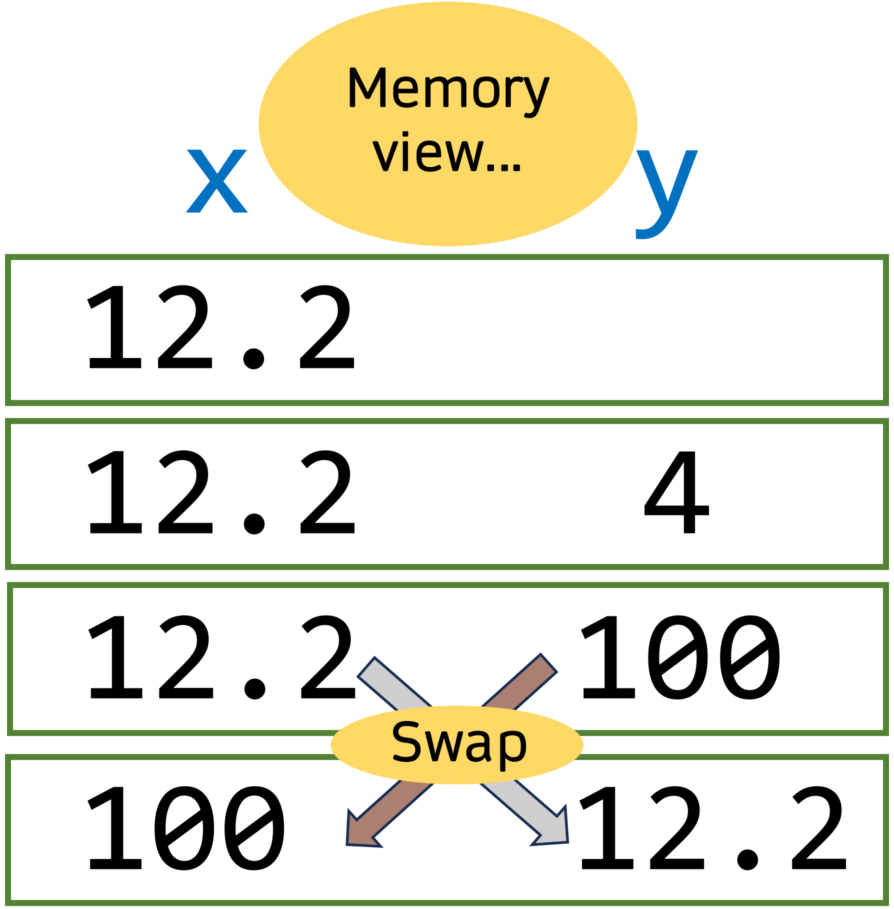
Will this code run?



Variables and Assignment Statements

Program view...

```
x = 12.2
y = 4
y = 100
x,y = y,x
```



Memory view...

x	y
12.2	
12.2	4
12.2	100
100	12.2

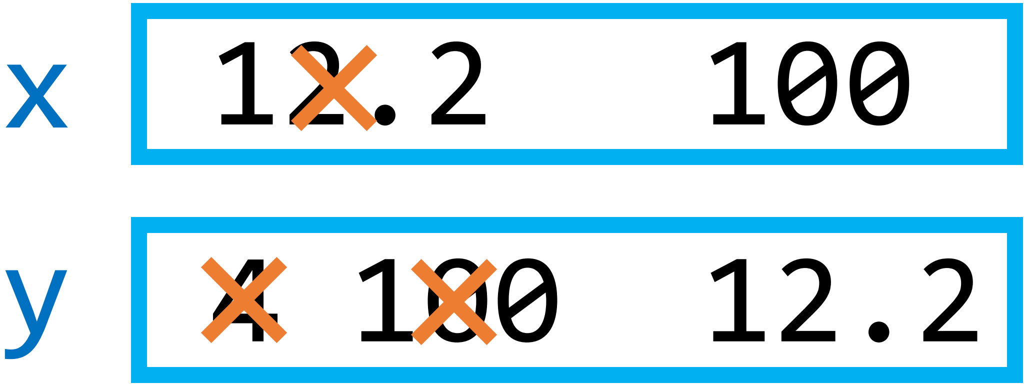
The assignment operator '**=**' has the format
<dest/store/LHS> = <source/load/RHS>
RHS is always evaluated (executed) first...

Variables and Assignment Statements

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right-hand side (Rval) and a variable to store the result in the left-hand side (Lval)

```
x = 12.2
y = 4
y = 100
temp = x
x = y
y = temp
```

Swapping without using 3rd variable is another way to do it.



x ~~12.2~~ 100

y ~~4~~ ~~100~~ 12.2

An aside: round(x) at half-way => x+1, or x-1?

```
>>> round(-0.5)

>>> round(0.5)

>>> round(1.5)

>>> round(2.5)

>>> round(3.5)
```

What will be the output of these statements?

```
>>> round(-0.5)
0
>>> round(0.5)
0
>>> round(1.5)
2
>>> round(2.5)
2
>>> round(3.5)
4
```

For equally close cases, rounding is done toward the even choice…

Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive

```
Good:    spam    eggs   spam23    _speed
Bad:     23spam     #sign  var.12
Different:    spam   Spam   SPAM
```

Mnemonic Variable Names

- Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”
- We name variables to help us remember what we intend to store in them (“mnemonic” = “memory aid”)
- This can confuse beginning students because well-named variables often “sound” so good that they must be keywords

http://en.wikipedia.org/wiki/Mnemonic

Mnemonic Variable Names

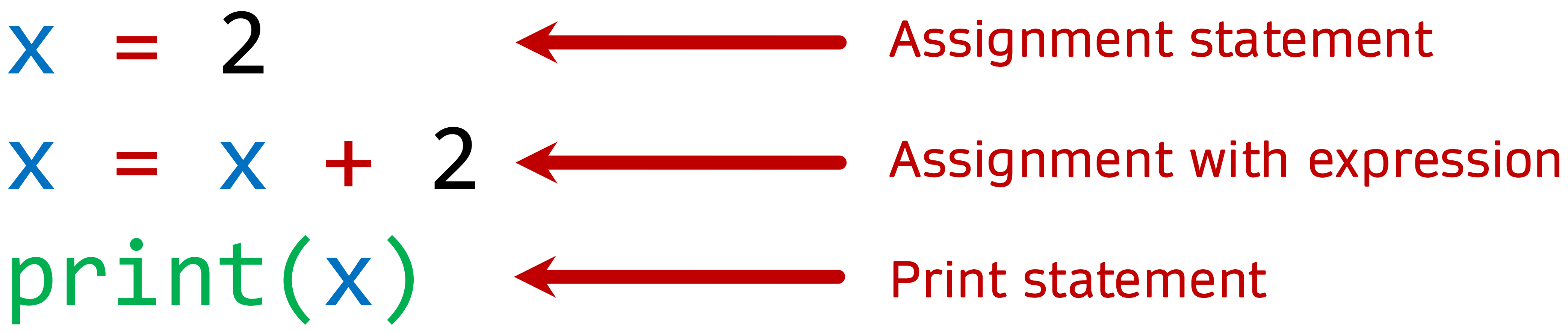
```
xxxabshghsjgjs = float(input())
prerajulisation = float(input())
yadharkarnathsha = float(input())
farhanitrate = xxxabshghsjgjs * prerajulisation * yadharkarnathsha
print(farhanitrate/100)
```

```
a = float(input())
b = float(input())
c = float(input())
d = a * b * c
print(d/100)
```

```
principal = float(input())
rate = float(input())
time = float(input())
simple_interest = principal * rate * time
print(simple_interest/100)
```

What are these codes doing?

Revisiting: Sentences or Lines



```
x = 2
x = x + 2
print(x)
```

Assignment statement

Assignment with expression

Print statement

Variable Operator Constant Function

Expressions: Numeric Expressions

- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) looks different than in math

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Expressions: Numeric Expressions

```
>>> x = 1
>>> print(x)
1
>>> x = x + 1
>>> print(x)
2
>>> x = 5
>>> y1 = x % 2
>>> y2 = x / 2
>>> y3 = x // 2
>>> print(y1,y2,y3)
1 2.5 2
```

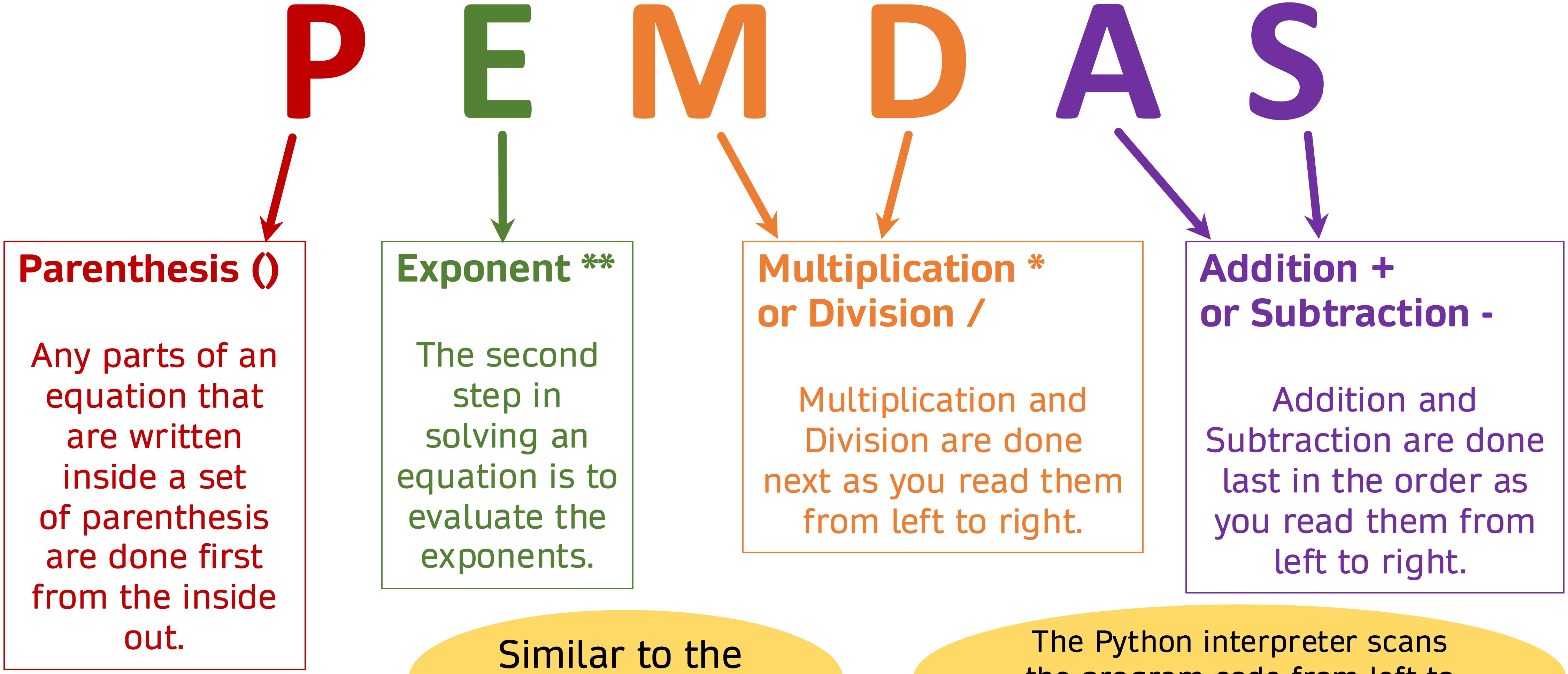
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Order of Evaluation – operator precedence

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”
- Which operator “takes precedence” over the others?

```
x = 1 + 2 * 3 - 4 / 5 ** 6
```

Order of Evaluation – operator precedence



Parenthesis ()

Any parts of an equation that are written inside a set of parenthesis are done first from the inside out.

Exponent **

The second step in solving an equation is to evaluate the exponents.

Multiplication * or Division /

Multiplication and Division are done next as you read them from left to right.

Addition + or Subtraction -

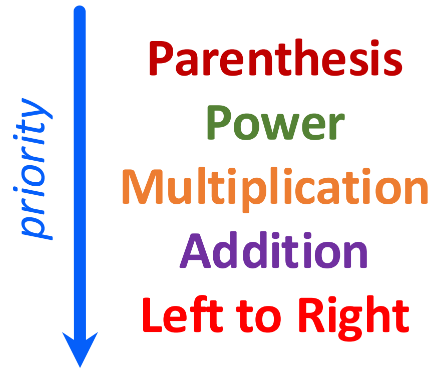
Addition and Subtraction are done last in the order as you read them from left to right.

Similar to the BODMAS rule?

The Python interpreter scans the program code from left to right in that order…

Operator Precedence Rules

- Highest precedence rule to lowest precedence rule:
  - Parentheses are always respected
  - Exponentiation (raise to a power)
  - Multiplication, Division, and Remainder (%)
  - Addition and Subtraction
  - Left to right



The Python interpreter scans the program code from **left to right** in that order…

Left to right also acts as a tie-breaker for the same precedence level (except exponentiation and conditionals) to resolve ambiguity (in the absence of parenthesis)…

Example: operator precedence

```
x = 1 + 2 * 3 - 4 / 5 ** 6
x = 1 + 2 * 3 - 4 / 5 ** 6
x = 1 + 2 * 3 - 4 / 15625
x = 1 + 2 * 3 - 4 / 15625
x = 1 + 6 - 4 / 15625
x = 1 + 6 - 4 / 15625
x = 1 + 6 - 0.000256
x = 7 - 0.000256 = 6.999744
```

priority

Parenthesis

Power

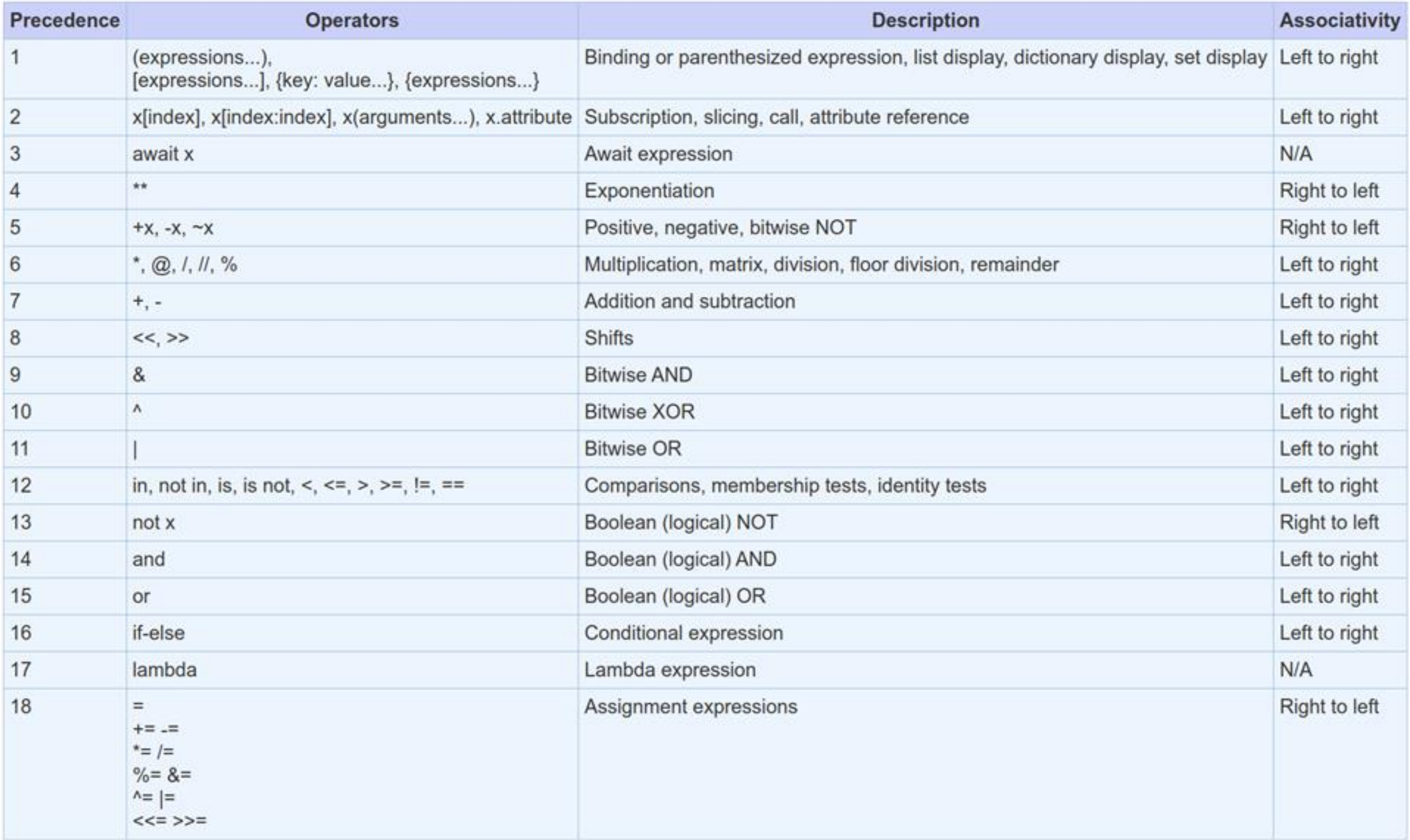
Multiplication

Addition

Left to Right

Operator Precedence in Python

priority



Precedence	Operators	Description	Associativity
1	(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or parenthesized expression, list display, dictionary display, set display	Left to right
2	x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference	Left to right
3	await x	Await expression	N/A
4	**	Exponentiation	Right to left
5	+x, -x, ~x	Positive, negative, bitwise NOT	Right to left
6	*, @, /, //, %	Multiplication, matrix, division, floor division, remainder	Left to right
7	+, -	Addition and subtraction	Left to right
8	<<, >>	Shifts	Left to right
9	&	Bitwise AND	Left to right
10	^	Bitwise XOR	Left to right
11	|	Bitwise OR	Left to right
12	in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, membership tests, identity tests	Left to right
13	not x	Boolean (logical) NOT	Right to left
14	and	Boolean (logical) AND	Left to right
15	or	Boolean (logical) OR	Left to right
16	if-else	Conditional expression	Left to right
17	lambda	Lambda expression	N/A
18	= += -= *= /= %= &= ^= |= <<= >>=	Assignment expressions	Right to left

Operator Precedence in Python

Operator	Description
(expressions...), [expressions...]	Binding or parenthesized expression, list indexing using expressions

```
>>> x=2
>>> y=6
>>> z=x-y*x+y
>>> z
-4
>>> z=(x-y)*x+y
>>> z
-2
```

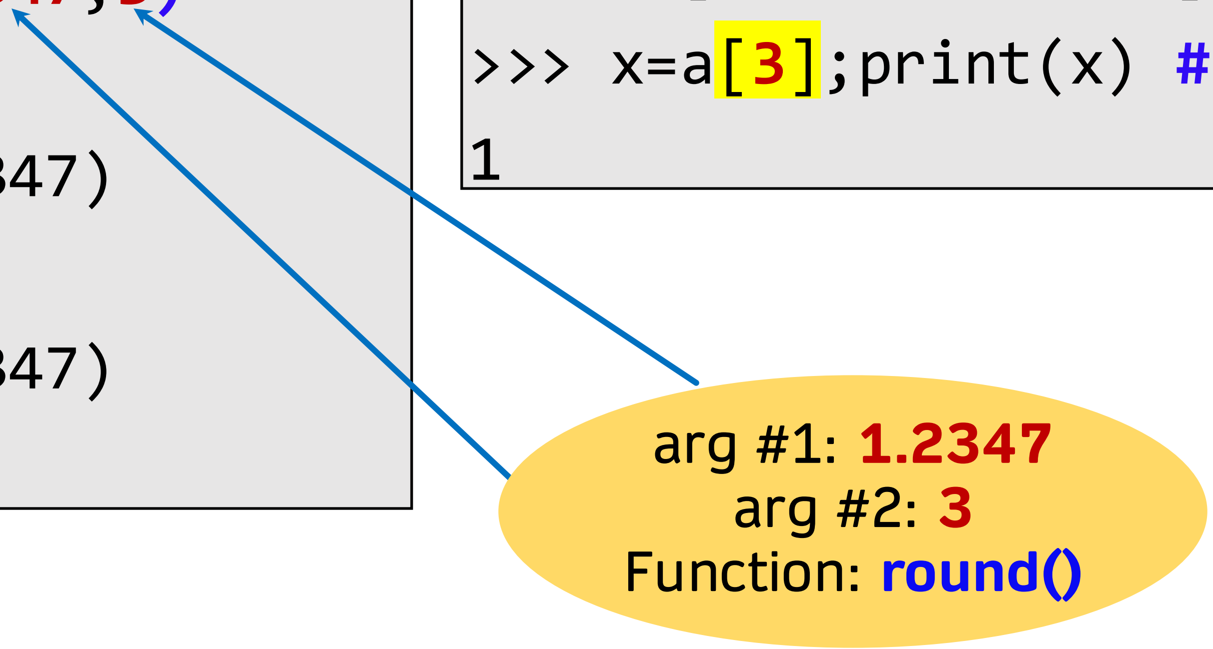
```
>>> z=(x-y)*(x+y)
>>> z
-32
>>> a=[1,2,3,4,5,6,7] #this is a list
>>> x=a[0];print(x) #zeroth element
1
>>> x=a[2*5-5] #6th element, i.e., a[5]
>>> x
6
```

Operator Precedence in Python

Operator	Description
x[index], x(arguments...)	List indexing when **index value is known**, function call

```
>>> round(1.2347,3)
1.235
>>> round(1.2347)
1
>>> round(1.7347)
2
```

```
>>> a=[1,2,3,4,5,6,7] #this is a list
>>> x=a[3];print(x) #fourth element
1
```



arg #1: **1.2347**
arg #2: **3**
Function: **round()**

Operator Precedence in Python

Operator	Description
**	Exponentiation (raised to the power)

```
>>> 2**3
8
>>> 2**3**4
2417...12352 #suppressed
>>> (2**3)**4
4096
>>> 2**-2
0.25
```

Precedence of **: right to left. First 3**4 is evaluated. Call this k, then 2**k is evaluated.

```
>>> 2**0.5
1.4142135623730951
>>> -2**0.5
-1.4142135623730951
>>> (-2)**0.5 #this will be a complex no.
(8.659560562354934e-17+1.4142135623730951j)
```

Operator Precedence in Python

Operator	Description
`+x, -x, ~x`	Positive (unary), negative (unary), bitwise NOT (unary)

```
>>> x=8 #this is 8 in decimal (base 10)
>>> bin(x)
'0b1000' #this is 8 in binary (base 2)
>>> int(0b1000) #'' are only for display, not in binary form
8
>>> int(bin(x))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '0b1000'
```

Operator Precedence in Python

Operator	Description
+x, -x, ~x	Positive, negative, bitwise NOT

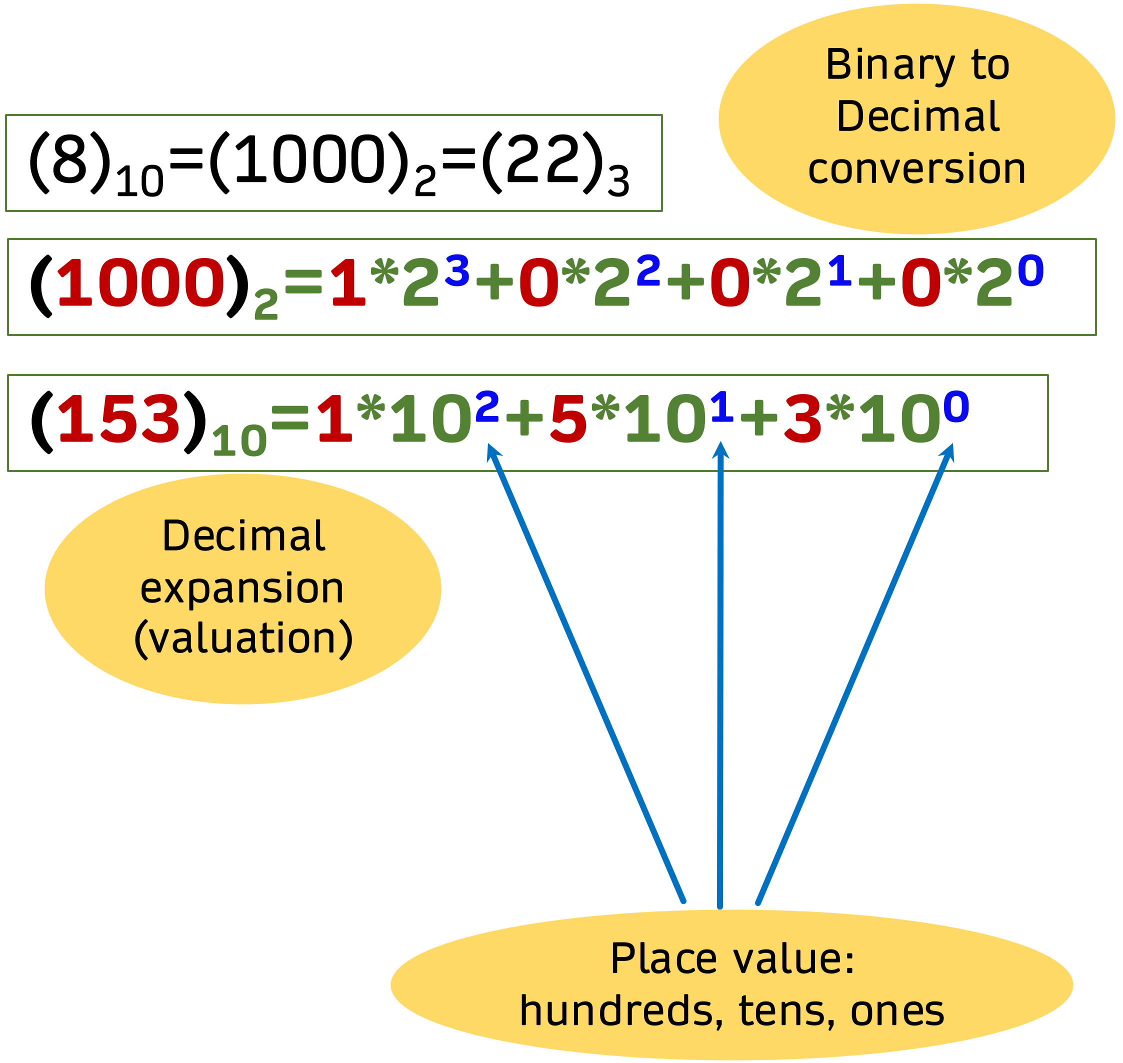
```
>>> x=8 #this is 8 in decimal (base 10)
>>> +x  #positive x
8
>>> -x  #negative x
-8
>>> ~x  #bitwise NOT of x equals to -(x+1) in base 10
-9
```

In case of binary system (base 2), negative numbers are in 2's complement form.

In 2's complement notation, left most bit is the sign bit, and the rest is magnitude.

Operator Precedence in Python

(base 10)	(base 2)	(base 3)
0	0	0
1	1	1
2	10	2
3	11	10
4	100	11
5	101	12
6	110	20
7	111	21
8	1000	22



$(8)_{10}=(1000)_2=(22)_3$

Binary to Decimal conversion

$(1000)_2=1*2^3+0*2^2+0*2^1+0*2^0$

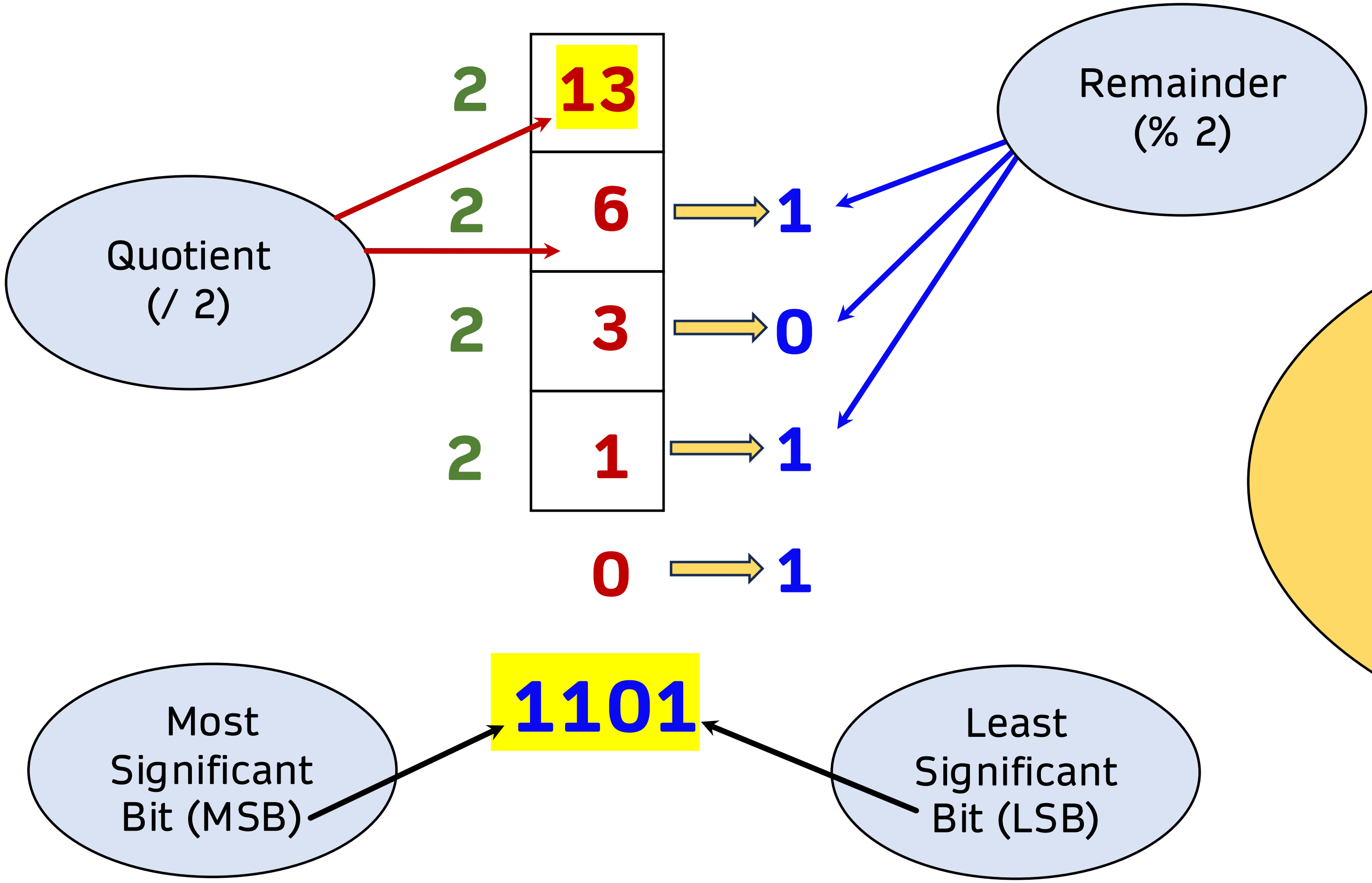
$(153)_{10}=1*10^2+5*10^1+3*10^0$

Decimal expansion (valuation)

Place value: hundreds, tens, ones

Operator Precedence in Python

Decimal to Binary conversion



Divide the number (dividend) repeatedly by 2 (divisor) until the dividend collapses to 0 (zero). In each step note down the remainder (0 or 1). The remainders written down from left to right in the reverse order, is the original number's binary representation.

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Contributors 2024 - Yogesh K. Meena and Shouvick Mondal, IIT Gandhinagar