Information, Computation, Communication

Learning Python

Numbers and Booleans

Agenda

- Numbers
- Booleans
- Comparisons
- Logical operations with other types
- Assignments
- Operator precedence

Next topic: if-elif-else

Numbers and Types

- Python has three **numeric types**:
  - `int`: integer
    - `1, 2, 3, …`
    - `-1, -2, -3` …
  - `float`: floating-point numbers
    - `0.123, 109.239292,` …
  - `complex`: complex numbers
    - `1j`
    - `3 + 5j`

Arithmetic Operations

- Type below numbers and operations in the Python interpreter and observe the output (i.e., the response after pressing **`Enter`**)

```
3                   # => 3 (an integer number)
3.0                 # => 3.0 (a floating number)
1 + 1               # => addition, returns 2
8 - 1               # => subtraction, returns 7
(1+2j) * 2          # => multiplication with complex
                    # numbers, returns 2 + 4j (complex)
```

Arithmetic Operations

Division

- Python supports two division operators: true and integer division

```
5 / 2     # => true division, 2.5
13 / 4    # => true division, 3.25
9 / 3     # => true division, 3.0
7 / 1.4   # => true division, 5.0

5 // 2    # => floor (also called integer) division, 2
13 // 4   # => floor division, 3
7 // 1.4  # => floor division, 5.0
-10 // 3   # => -4
```

Arithmetic Operations

Integer division

To remember

- True division always returns a floating point number
- In integer division, the result type depends on the types of the operands:
  - int // int => int
  - float // float => float
  - One integer and one float => float
- To change the numeric type, we use built-in functions int( ), float( )
  - The type change operation is also called type casting
- Integer division rounds fractional remainders down
  - Regardless of type and sign
  - The result is always lower or equal to the result of the true division

Arithmetic Operations

Exponentiation and modulus

```
2 ** 4    # => exponentiation, 2**4 = 2*2*2*2 = 16

5 % 2     # => integer remainder (modulus), returns 1
          # 5 % 2 = 5 - 2 * (5//2) = 5 - 4 = 1

9 % 3     # => integer remainder (modulus), returns 0

-10 % 3   # => modulus, returns 2 !!!
          # -10 % 3 = -10 - 3 * (-10//3) = -10 - 3*(-4)
```

Booleans

Booleans

- Bool type (Boolean) is a subtype of integer
- True equals 1, and False equals 0
- Boolean types are common in control flow expressions (if-else, loop)

```
True              # => returns True
False             # => returns False
type(True)        # => <class 'bool'>
True + 4          # => returns 5
```

Note the capital first letter:

- True
-  true
- False
-  false

Boolean (Logical) Operations

```
x or y    # => logical OR
```

Logical OR:
True or ??? => True
False or True => True
False or False => False

```
x and y   # => logical AND
```

Logical AND:
True and True => True
True and False => False
False and ??? => False

```
not x     # => logical negation
```

Logical negation:
not True => False
not False => True

Comparisons

Also known as relational operators

Comparisons

```
1 < 2              # => Evaluate if 1 is less than 2,
                   # returns True
2.0 >= 1           # => Evaluate if 2.0 is greater than or
                   # equal to 1, returns True
7 == 7             # => Evaluate if 7 equals 7,
                   # returns True
True == 1          # => Evaluate if True equals 1,
                   # returns True
2.0 != 2           # => Evaluate if 2.0 is different than
                   # 2, returns False
                   # (2.0) != type(2) would return True
```

Comparisons: Chained

```
x = 2  # create integer variable x and assign 2 to it
y = 4  # create integer variable y and assign 4 to it
z = 6  # create integer variable z and assign 6 to it
# Use variables in expressions
x < y < z      # => x < y  and y < z, returns True

x < y > z      # => x < y  and y > z, returns False

x == y < z     # => x == y and y < z, returns False
```

Logical Operations with Types other than Boolean

Boolean (Logical) Operations

In the context of Boolean operations (and, or, not),
the following values are interpreted as False:

- False
- None (a value commonly used to signify ‘empty’, or ‘no value’)
- Numeric zero of all numeric types
- Empty strings
- Empty lists, tuples, dictionaries, sets, …

All other values are interpreted as True

Logical Operations with Nonboolean Types

```
x or y    # => logical OR
# from left to right



# Examples:
3 or 5          # => 3
5 or 'EPFL'     # => 5
0 or 3          # => 3
False or 'EPFL' # => 'EPFL'
```

Logical OR, x or y:
If x is (interpreted as) True => x
If x is (interpreted as) False => y

Logical Operations with Nonboolean Types

```
x and y   # => logical AND
# from left to right
```

Logical AND, x and y:
If x is (interpreted as) True => y
If x is (interpreted as) False => x

```
# Examples:
3 and 5         # => 5
5 and 'EPFL'    # => 'EPFL'
0 and 3         # => 0
'EPFL' and False    # => False
```

Logical Operations with Nonboolean Types

```
not x     # => logical negation
# from right to left



# Examples:
not 3         # => False
not 0         # => True
not False     # => True
not 'EPFL'    # => False
```

Logical negation, not x:

If x is (interpreted as) True => False

If x is (interpreted as) False => True

Logical Operations with Nonboolean Types

Examples

What is the value of the following expressions?

```
# Example 1
temp = 17
result = ((temp > 15) and 'black dress') or 'jeans'
```

```
# Example 2
temp = 10
result = ((temp > 15) and 'black dress') or 'jeans'
```

Logical Operations with Nonboolean Types

Examples

What is the value of the following expressions?

```
# Example 1

temp = 17

result = ((temp > 15) and 'black dress') or 'jeans'

# answer: 'black dress'; # steps:
# (1) temp > 15 returns True
# (2) True and 'black dress' returns 'black dress'
# (3) 'black dress' or 'jeans' returns 'black dress'


# Example 2

temp = 10

result = ((temp > 15) and 'black dress') or 'jeans'
# answer: 'jeans'
```

Assignments

Assignment

Operator	Syntax	Meaning
=	x = y + z	Assign x = y + z
+=	x += y	Add and assign x = x + y
-=	x -= y	Subtract and assign x = x – y
*=	x *= y	Multiply and assign x = x * y
/=	x /= y	Divide (true) and assign x = x / y
%=	x %= y	Compute modulo and assign x = x % y
//=	x //= y	Divide (integer) and assign x = x // y
**=	x **= y	Calculate exponent and assign x = x ** y

- Assignment operator computes the value of the expression on the right and assigns it to the operand on the left
- Assignment operator can be combined with arithmetic operators

Assignment Operators

Examples

What is the value of the following expressions?

```
# Example 1
a = 5
x = 3
y = 0
a *= x - (y <= x)
```

EXAMPLES

Assignment Operators

Examples

EXAMPLES

What is the value of the following expressions?

```
# Example 1
a = 5
x = 3
y = 0
a *= x - (y <= x)
# answer: a = 10
# steps:
# (1) y <= x returns True
# (2) x - True returns 2
# (3) a *= 2  returns a*2 which equals 10
```

Operator Precedence

Precedence of Operators

- Python will always evaluate the arithmetic operators first
  - ** is highest, then multiplication/division, then addition/subtraction
- Next come the relational operators
- The logical operators are done last

Priority level	Category	Operators	Associativity
7 (highest)	Exponent	**	**right to left**
6	Multiplication, etc.	*, /, //, %	left to right
5	Addition and subtraction	+, -	left to right
4	Relational	<=, >=, >, <, ==, !=	left to right
3	Logical	not	**right to left**
2	Logical	and	left to right
1 (lowest)	Logical	or	left to right

Precedence of Operators

The acronym **PEMDAS** is a convenient way to remember the rules

- **P**arentheses have the **highest** precedence and can be used to force an expression to evaluate in the order you want
- **E**xponentiation has the next highest precedence
- **M**ultiplication and **D**ivision have the same precedence, which is **higher** than **A**ddition and **S**ubtraction, which also have the same precedence

Precedence of Operators

Examples - PEMDAS

What is the value of the following expressions?

```
# Example 1
result = True or False and False
```

EXAMPLES

Precedence of Operators

Examples - PEMDAS

What is the value of the following expressions?

```
# Example 1
result = True or False and False
# answer: True
# Steps:
#     and has a higher precedence over or
#     result = True or (False and False)
#     False and False => False
#     True or False   => True
```

Precedence of Operators

Examples - PEMDAS

What is the value of the following expressions?

```
# Example 2
result = 2 ** 3 ** 2
```

EXAMPLES

Precedence of Operators

Examples - PEMDAS

What is the value of the following expressions?

```
# Example 2
result = 2 ** 3 ** 2
# answer: 512
# Steps:
#    associativity: right to left
#    result = 2 ** (3 ** 2)
#    3 ** 2 = 9
#    result = 2 ** 9 = 512
```

EXAMPLES

Precedence of Operators

Examples - PEMDAS

What is the value of the following expressions?

```
# Example 3
a = b = 0
result = a < b + 5
```

Precedence of Operators

Examples - PEMDAS

What is the value of the following expressions?

```
# Example 3
a = b = 0
result = a < b + 5
# answer: True
# Steps:
#    addition has a higher precedence over <
#    result = 0 < (b + 5)
#    b + 5 = 5
#    0 < 5 => True
```

Next topic: If-elif-else