CS1302 - Lecture 3 Conditional Execution *(Conditional Execution.ipynb)*
==========

1 **Boolean expressions**
- Motivation
- Boolean expressions and Boolean data type

2 **Comparison operators**

2.1 **Comparison Operators (>, >=, ==, != etc.)**
- How to compare different values
- **Precedence** and **Associativity**
- Comparing **Different Data Types**
- Using ! in python notebooks
- Comparing **floating point numbers** and **absolute tolerance**
- Review **isclose**

2.2 **Boolean Operations (and, or, etc.)**
- Why use **boolean operators** (**logical operators**)
- **Truth Table**
- **Chained** logical operators
- **precedence** and **associativity** for the logical operators
- **Short-circuit evaluation**
- Logical operators on non-boolean operands

3 **Conditional Execution**
- **Conditional Expression see 3.2**

3.1 If-Then Construct
- *Sort 2 values*
- **Indentation**
- **pass** - Leave the block empty

3.2 If-Then-Else Construct
- if-statement with **else** clause
- **conditional expression**

3.3 Nested Conditionals
- *Sort 3 values*
- **nested if-statement with else clause**
- if-**elif**-else
- eliminate duplicate checks

==========

More and Cheat notes

==========

1 **Boolean expressions**

Motivation

```
# Problem (division by zero)
def multiply_or_divide(a, b):
    print("a, b, a*b, a/b are: ", a, b, a * b, a / b, sep="\n")
multiply_or_divide(1, 2)
multiply_or_divide(1, 0)


# Check and handle problem (I)
def multiply_or_divide(a, b):
    if b == 0:    # == means comparison: same?
        print("a, b, a*b, a/b are: ", a, b, a * b, "undefined", sep="\n")
    else:
        print("a, b, a*b, a/b are: ", a, b, a * b, a / b, sep="\n")

multiply_or_divide(1, 0)
multiply_or_divide(1, 2)

# Check and handle problem (I')
def multiply_or_divide(a, b):
    print("a/b: ", "undefined") if b==0 else print("a/b: ", a/b)

# Check and handle problem (I'')
def multiply_or_divide(a, b):
    print("a/b:", a/b) if b!=0 else print("a/b:", "undefined")
    # b!=0 means "b is not equal to zero?"

# Check and handle problem (I''')
def multiply_or_divide(a, b):
    print("a/b:", a/b) if b else print("a/b:", "undefined")
    # when we want true/false, but given a number b, then "b is non-zero" => true

# Check and handle problem (II)
def multiply_or_divide(a, b):
    fix = a/b if b else "undefined"
    print("a, b, a*b, a/b are: ", a, b, a * b, fix, sep="\n")

multiply_or_divide(1, 0)
multiply_or_divide(1, 2)
```

Boolean expressions and Boolean data type

2 **Comparison operators**

2.1 **Comparison Operators (>, >=, ==, != etc.)**

How to compare different values

Precedence and **Associativity**

https://docs.python.org/3/reference/expressions.html?highlight=precedence#operator-precedence

Canvas=>CS1302=>Piazza=>"[Lec02, 03] precedence and associativity of +,-,*,/,%,**,>,>=,<,<=,==,!=,and,or,not"

Comparing **Different Data Types**

Using ! in python notebooks

Comparing **floating point numbers** and **absolute tolerance**

About **isclose**

```
import math
x = 10
y = (x ** (1 / 2)) ** 2
x, y, math.isclose(x,y)

# About rel_tol default rel_tol is 1e-09
x, y, math.isclose(x,y, rel_tol= 1e-17)
x, y, math.isclose(x,y, rel_tol= 1e-09)
# rel_tol is to multiply with 10.0000000000002 => give the tolerance

math.isclose(0, 0.00000000000000000000000001)
# rel_tol is to multiply with 0.00000000000000000000000001 => tolerance

# About abs_tol
math.isclose(0, 0.00000000000000000000000001, abs_tol=1e-09)

math.isclose?

# default rel_tol is 1e-09, abs_tol=0.0 (abs_tol is useful for comparisons near zero)
result1 = math.isclose(10, 10.000000001)    # True
result2 = math.isclose(10, 10.00000001)    # False
result3 = math.isclose(0, 0.00000000000000000000000001)    # False
result4 = math.isclose(0, 0.00000000000000000000000001, abs_tol=1e-9)    # True
result1, result2, result3, result4

# to conclude:
# For the values to be considered close, the difference between them must be smaller than at least
# one of the tolerances (applying rel_tol to the bigger magnitude of input values, applying abs_tol)
# Like this: abs(x - y) <= max(rel_tol * max(abs(x), abs(y)), abs_tol)
```

2.2 **Boolean Operations (and, or, etc.)**

Why use **boolean operators** (**logical operators**)

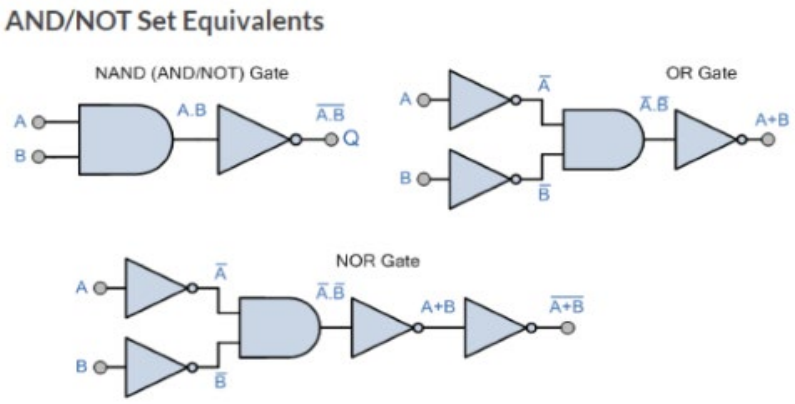
https://en.wikipedia.org/wiki/Functional_completeness

Below is for interested students only

https://www.electronics-tutorials.ws/logic/universal-gates.html

{AND} alone is not functionally complate

but
{AND, NOT} is functionally complete
we can use AND and NOT gates to make up other logic gates



Truth Table

Chained logical operators **(See Lecture03_Exercises)**

precedence and **associativity** for the logical operators **(See Lecture03_Exercises)**

Short-circuit evaluation (See Lecture03_Exercises)

Logical operators on non-boolean operands **(Also See Lecture03_Exercises)**

Operators, Truth Table, Non-boolean Operands, Evaluation with Short-circuit evaluation

Operator	Description	Example
and	Return True when both statements are true	x < 5 and x < 10
or	Returns True when one of the statement is true	x < 5 or x < 10
not	Reverse the result, return True when the statement is False	not (x < 5)

x	y	x and y	x or y	not x
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

The above table is called a *truth table*. It enumerates all possible input and output combinations for each boolean operator.

Indeed, logical operators can even be applied to non-boolean operands. From the documentation:

> In the context of Boolean operations, and also when expressions are used by control flow statements, the following values are interpreted as false: `False`, `None`, numeric zero of all types, and empty strings and containers (including strings, tuples, lists, dictionaries, sets and frozensets).
> All other values are interpreted as true.

Important

> The expression `x or y` first evaluates `x`;
> if `x` is true, its value is returned;
> otherwise, `y` is evaluated and the resulting value is returned.

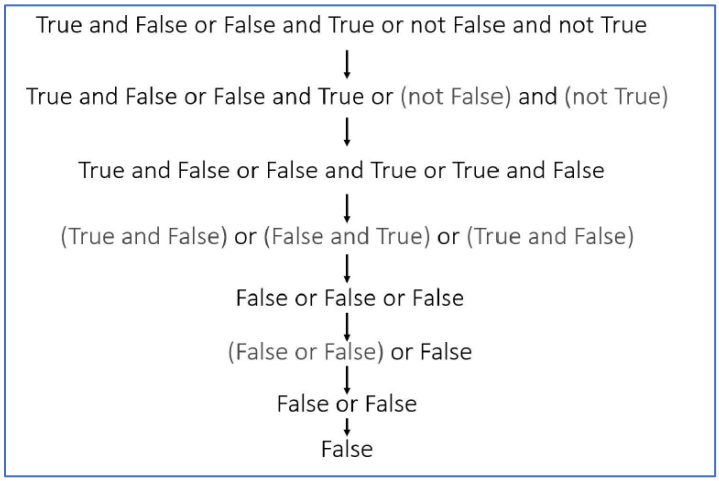
Important

> The expression `x and y` first evaluates `x`;
> if `x` is false, its value is returned;
> otherwise, `y` is evaluated and the resulting value is returned.

We know that number `0` means logical `False` and other numbers means logical `True`

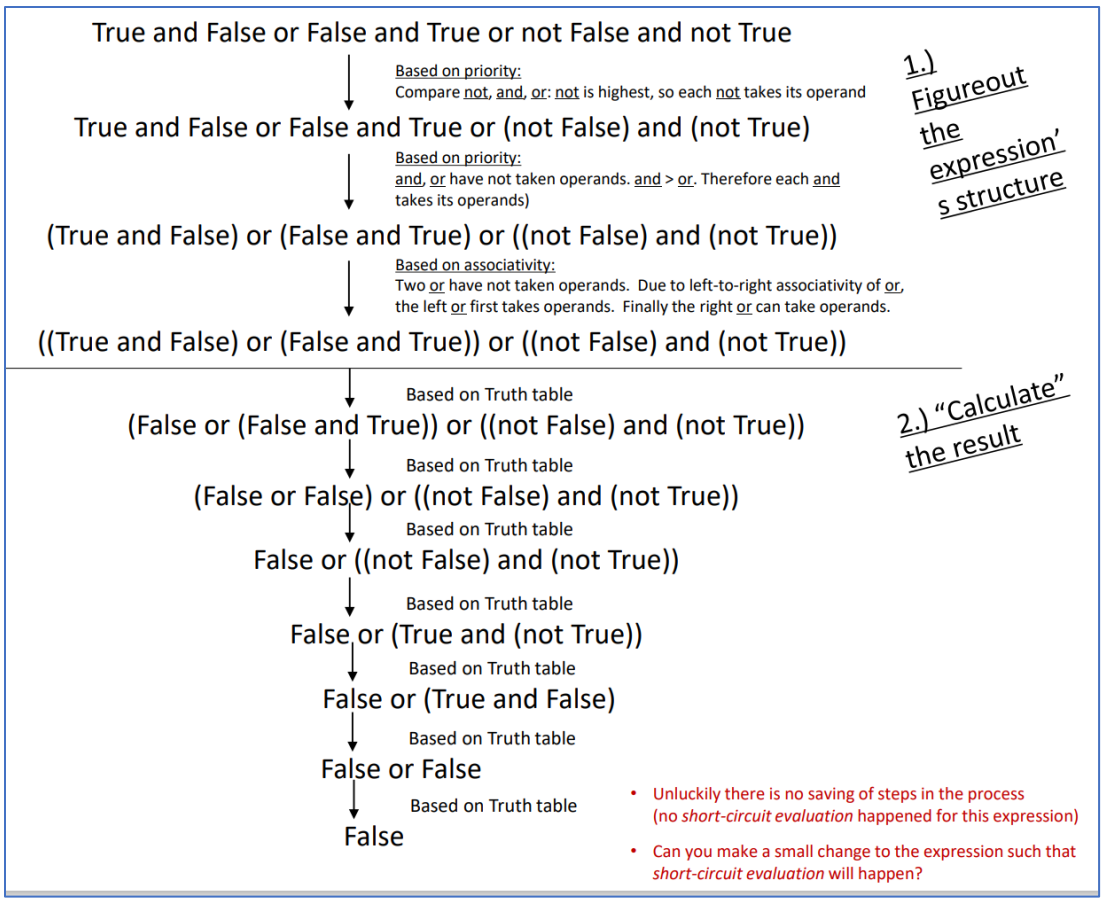
x	y	x and y	x or y	not x
not 0	doesn't matter	y	x	False
0	doesn't matter	x	y	True

The picture below is wrong:



The mistake: "(not False)" and "(not True)" should not be evaluated at the beginning.

The correct steps should be:



Testing

```
def returnFalse(label):
    print(label)
    return False

def returnTrue(label):
    print(label)
    return True

result1 = True and False or False and True or not False and not True
print(result1) # it shows False

result2 = returnTrue("a") and False or False and True or not returnFalse("b") and not returnTrue("c")
print(result2) # it shows a first: a b c False
```

To summarize:

Precedence and associativity of +,-,*,/,%,**,>,>=,<,<=,==,!=,and,or,not

[Question]
What is meant by **precedence** of operators of arithmetic operations (Lec02), and comparisons and logical operations (Lec03)?

What does it apply to? How about **associativity** (Lec02)?

[Answer]
See the table below.
It lists some operators according to their ***precedence*** (~*priority*).
Upper ones have higher precedence than the lower ones.
(The contents are selected from https://docs.python.org/3/reference/expressions.html#operator-precedence)

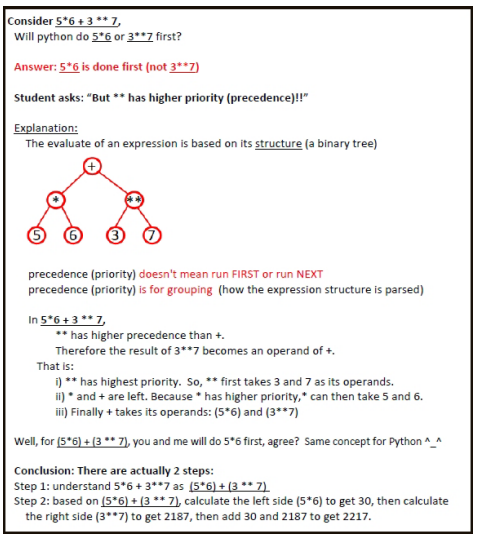
Some common python operators in **descending order** of **Precedence**

Operator	Description
`**`	Exponentiation
`-x`	negative
`*`, `/`, `//`, `%`	Multiplication, division, floor division, remainder
`+`, `-`	Addition and subtraction
`<`, `<=`, `>`, `>=`, `!=`, `==`	Comparisons and identity tests
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`:=`	Assignment expression

Note:

Operator precedence
Determines how operators are parsed concerning each other. Operators with higher precedence become the operands of operators with lower precedence.
(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence)

See this SIMPLE example on the right👍:



Associativity:
When two operators have the same precedence, **associativity** helps to determine the order of considerations.
(https://www.programiz.com/python-programming/precedence-associativity)

For binary arithmetic operators (+, -, *, /, %, ** ..) :

precedence: ** > * and / and % etc. > +, -

associativity: ** is right-to-left; others are left-to-right
e.g. **a**b**c** means **a ** (b**c)**
e.g. **a+b-c** means **(a+b) - c**

For comparison operators (<, <=, >, >=, !=, ==) :

precedence: all are the same, and all are below + and -

associativity: non-associative
e.g. **a>b==c** means **(a>b) and (b==c)**

For logical operators (or we say "Boolean operators": and, or, not) :

precedence: not > and > or

associativity: left-to-right
e.g. **a and b and c** means **(a and b) and c**

For chained logical expression, **truth table**, **short-circuit evaluation**:

Evaluating chained Boolean expression

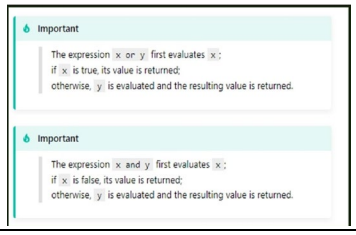
Stage 1.) Figure out the expression's structure
- Priority: not > and > or
- Associativity: left associative (A and B and C)

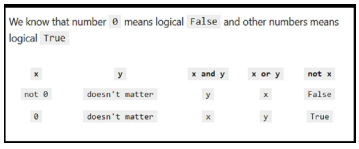
Stage 2.) "Calculate" the result
- Truth table:

x	y	x and y	x or y	not x
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

- Short-circuit evaluation:
  - True or B or C or D or E or... must be True
  - False and B and C and D and E and ... must be False

For involving non-boolean operands:





[To see explanations and examples, please refer to Lecture 2 and Lecture 3's *notebooks (.ipynb)*]

---

3 **Conditional Execution**

Conditional Expression see 3.2

3.1 If-Then Construct

Sort 2 values

Indentation

pass - Leave the block empty

3.2 If-Then-Else Construct

if-statement with **else** clause

conditional expression

3.3 Nested Conditionals

Sort 3 values

nested if-statement with else clause

if-**elif**-else

eliminate duplicate checks

```
==============
# if
x, y = 25, 25
if x <= y:
        print(x, y)
if y < x:
        print(y, x)

# "flow chart" (see lecture notebook: sort_two_values_fc1)

==============
# if
x, y = 25, 25
if x <= y: print(x, y)
if y < x: print(y, x)
==============
elderly = True
alone = True
disabled = True

# send a gift for elderly, or alone and disabled
gift = ____
if ____:
        gift = ____
if ____:
        if ____:
                gift = ____

print(gift) # expect True/False
==============
```

```
# if-else
x, y = 25, 25
if x <= y:
	print(x, y)
else:
	print(y, x)
==============
# if-else-pass
x, y = 25, 25

if x == y:
	pass    # not yet decided what to output (leave this block empty)
else:
	if x < y:
		print(x, y)
	else:
		print(y, x)
==============
# if-else
x, y = 25, 25

if x == y:
	print("same:", x)
else:
	if x < y:
		print(x, y)
	else:
		print(y, x)
==============
# elif

x, y = 25, 25

if x == y:
	print("same:", x)
elif x < y:
	print(x, y)
else:
	print(y, x)
================
# conditional expression
x, y = 40, 50

output1 = str(x) + " " + str(y)
output2 = str(y) + " " + str(x)

s = output1 if x <= y else output2
print(s)
===============
```

```
# conditional expression (ERROR 1!)
x, y = 40, 50

output1 = str(x) + " " + str(y)
output2 = str(y) + " " + str(x)

s = output1 if x <= y
print(s)
===============
# conditional expression (ERROR 2!)
x, y = 40, 50

output1 = str(x) + " " + str(y)
output2 = str(y) + " " + str(x)

s = output1 if x <= y else s = output2
print(s)
=============
elderly = True
alone = True
disabled = True

# send a gift for elderly, or alone and disabled
gift = _____ (No logical operators: and, or, not)

print(gift) # expect True/False

#Hint: gift = True if elderly else False if alone==False else True if disabled else False
=============

# sort 3 numbers
x, y, z = 10, 50, 1

if x <= y <= z:      #this condition checks (x<=y) and (y<=z)
      print(x, y, z)
elif x <= z <= y:
      print(x, z, y)
elif y <= x <= z:
      print(y, x, z)
elif y <= z <= x:
      print(y, z, x)
elif z <= x <= y:    #when we check this condition (z<=x) and (x<=y), actually the second part has already
      print(z, x, y) #been checked in the first condition
else:
      print(z, y, x)
===============
```

```
# sort 3 numbers (more efficient)
x,y,z=10,50,1

if x <= y:
    if y <= z:
        print(x, y, z)
    elif x <= z:
        print(x, z, y)
    else:
        print(z, x, y)
elif z <= y:
    print(z, y, x)
elif z <= x:
    print(y, z, x)
else:
    print(y, x, z)
```

==========

Self-study:

- go through the **lecture notebook** and **Readings**
- do EQuiz 3 (Available next Monday)