CS1302 - Lecture 3 Conditional Execution

(Dr Helena WONG)

==========

Warm up and Break-time Exercises:

Try the following questions

==========

==========

Warm up Exercises

==========

Q1: 20+16*4 gives _____

Q2: 2**3**2 gives _____

Q3: 20-16+4 gives _____

Q4: int(True) gives _____

int(False) gives _____

Q5: Try and explain: x=y=1

Try and explain: x=(y=1)

Q6: The following code uses print and {} and format. Try!

```
year = 2025
print('This year is {}, next year is {}'.format(year, year+1))
# can change to use print(..+..+..+..) or print(..,..,..,..)?
# using .format, sep and end?
```

==========

Break-time Exercise A

123 > 45, "123" > "45", "big" > "small"

==========

Q1: 123>45 gives ____

Q2: "123">"45" gives ___

Q3: "big" > "small" gives ____

==========

Break-time Exercise B

==========

Q1: First, please set x and y:

```
x,y = 30,20
```

Now, evaluate the output of the following

```
x>y, y>10
```

Is the result reasonable to you?

Q2 Repeat Q1 for: x>y and y>10

Q3 Repeat Q1 for: x > y > 10

Q4 Repeat Q1 for: (x>y) > 10

==========

Break-time Exercise C

==========

Given that x and y hold some values.

Why `(x>y)>1` always give False

==========

Break-time Exercise D

==========

Recall: Precedance and Associativity

[Review]

Q1: `20+16*4` means? (20+16)*4? 20+(16*4)?

Q2: `2**3**2` means? (2**3)**2? 2**(3**2)?

Q3: `20-16+4` means? (20-16)+4? 20-(16+4)?

[Given]

A op B op C:

Left-assoiciativity is (A op B) op C

Right-assoiciativity is A op (B op C)

Q4: What is the **precedence** of comparison operators (>, ==, etc.)?

Q5: What is the **associativity** of comparison operators (>, ==, etc.)?

* All the comparison operators have the same precedence lower than that of + and -.
* Non-associative!

Q6: How about boolean (logical) operators (and, or, not)?

==========

Break-time Exercise E

==========

Try the code below:

Q1 True and "Good"

Q2 False and "Good"

Q3 True or "Good"

Q4 False or "Good"

Note: You may get True or False or Good

So, how about **4 and 5**? What is the result?

Q5 False == False

Q6 bool("")

Q7 bool("") == False

Q8 "" == False

An empty string is regarded as false in a boolean operation (when expect True/False) but a comparison operation (==) is not a boolean operation, even though it forms a boolean expression.

==========

Break-time Exercise F

==========

Using And, Or in daily life

What do the following mean?

(a) If you get average at least 80 and you have taken at least 3 courses, then good job!

(b) if the passenger is sick or i am young and he is old , then I give out my seat.

==========

Cheat notes

==========

Warm up Exercises

```
20+16*4, 2**3**2, 20-16+4, int(True), int(False)
```

Q1: 20+16*4 means? (20+16)*4? 20+(16*4)? #due to **precedence**

Q2: 2**3**2 means? (2**3)**2? 2**(3**2)? #due to **assoiciativity**

Q3: 20-16+4 means? (20-16)+4? 20-(16+4)? #due to **assoiciativity**

[Recall]

A op B op C:

Left-assoiciativity is (A op B) op C

Right-assoiciativity is A op (B op C)

Q4: int(True) gives 1 # will explain in Lec 3

int(False) gives 0 # will explain in Lec 3

```
"123", int("123"), type("123"), type(int("123"))

type(1), type(1302), type(-100), type(True), type(False)
# int: a lot of values (0,1,2,-100, 1302, ..)
# bool: two values only (True and False)
```

Q4': bool(1) gives **True**

bool(0) gives **False**

```
# int evaluted to boolean values:
#       0 => False
#       others => True
bool(0), bool(1), bool(-100)

# Strings evaluted to boolean values:
#      empty string => False
#      others => True
bool(""), bool("Hello"), bool("True"), bool("False")
```

Q5: Try and explain: x=y=1 #chained assignment

Try and explain: x=(y=1)

with "=", it is a statement (to execute)

it is not an expression (to get evaluated to a resultant value)

Q6: The following code uses print and {} and format. Try!

```
year = 2025
print('This year is {}, next year is {}'.format(year, year+1))
# can change to use print(..+..+..+..) or print(..,..,..,..)?
# use of "sep" and "end"
```

```
year = 2025
print('This year is {}, next year is {}'.format(year, year+1))

year = "2025"
print('This year is ' + year)

year = 2025
print('This year is ' + year)

year = 2025
print('This year is ' + str(year))

year = 2025
print('This year is', year) # no need the same type, default separator is a space

year = 2025
print('This year is', year) # default end is new line
print('Next year is', year+1)

year = 2025
print('This year is', year, end="; ") # default end is new line
print('Next year is', year+1)

print(year, year+1, year+2, sep="=>") # default separator is space

str1 = "here are our 3 favorites {0}, {1}, {2}"
print(str1.format(9.99999, 8, "eating"))

str1 = "here are our 3 favorites {2}, {0}, {1}, first is {0}"
print(str1.format(9.9999, 8, "eating"))

str1 = "here are our 3 favorites {c}, {b}, {a}"
print(str1.format(a=9.99999, b=8, c="eating"))

str1 = "here are our 3 favorites {fl}, {lucky_num}, {activity}, first is {activity}"
print(str1.format(fl = 9.9999, lucky_num=8, activity="eating"))
```

```
# f-string (for interested students only)
# https://docs.python.org/3/tutorial/inputoutput.html#tut-f-strings

# introducing f-string: a simplier way to do .format()
x = 6
str1 = 'you get {} marks'.format(x)
str2 = f'you get {x} marks'
print(str1, str2, sep='\n')  #str1 and str2 are the same

align, width = '^', 5
str3 = f':*{align}{width}' # str3 is a string ':*^5'
str4 = f'{{:*{align}{width}}}' # str4 is a string '{:*^5}'
# To have '{', we type '{{'
# To have '}', we type '}}'

print(str3)
print(str4) #str4 is just to add '{}' to str3

x = 6
str5 = '{:*^5}'.format(x)
str6 = str4.format(x)
print(str5)
print(str6) # str5 and str6 are the same

x, align, width = 6, '^', 5
str7 = f'{{:*{align}{width}}}'.format(x)
print(str7)
print('{:*^5}'.format(x))  # same as print(str7)
---------------
```

==========

Break-time Exercise A

123 > 45, "123" > "45", "big" > "small"

==========

Q1: 123>45 gives ____

Q2: "123">"45" gives ___

Q3: "big" > "small" gives ____

ASCII code of "b" is smaller

"bigger" > "biggest"

==========

Break-time Exercise B

==========

Q1: First, please set x and y:

x,y = 30,20

Now, evaluate the output of the following

x>y, y>10

Is the result reasonable to you?

Q2 Repeat Q1 for: x>y and y>10

No surprise

Q3 Repeat Q1 for: x > y > 10

(x>y) and (y>10)? (x>y)>10?

Q4 Repeat Q1 for: (x>y) > 10

True is greater than 10?

Note: It is like int(**True**) is greater than 10?

==========

Break-time Exercise C

==========

Given that x and y hold some values.

Why (x>y)>1 always give False

x, y = 20, 30

(x>y) > 1 # False > 1 (or: int(False)>1)

x, y = 30, 20

(x>y) > 1 # True > 1 (or: int(True)>1)

==========

Break-time Exercise D

==========

Recall: Precedance and Associativity

[Review]

Q1: `20+16*4` means? (20+16)*4? 20+(16*4)? #due to **precedence**

Q2: `2**3**2` means? (2**3)**2? 2**(3**2)? #due to **assoiciativity**

Q3: `20-16+4` means? (20-16)+4? 20-(16+4)? #due to **assoiciativity**

[Recall]

A op B op C:

Left-assoiciativity is (A op B) op C

Right-assoiciativity is A op (B op C)

[comparison operators] >, >=, ==, != etc.

Q4: What is the **precedence** of comparison operators (>, ==, etc.)?

All the comparison operators have the same precedence

They are lower than that of + and -.

Q5: What is the **associativity** of comparison operators (>, ==, etc.)?

Non-associative!

Note it's not evaluated from left to right (or right-to-left) directly.

Instead each comparison operation represent one boolean value

E.g., x<=y<=z means (x<=y) and (y<=z)

[boolean/logical operators] And, Or, Not

Q6: How about boolean (logical) operators (and, or, not)?

All binary boolean operators are left associative.

Precedence: comparison operators > not > and > or

Chained comparison operators and compound boolean expressions

Chained Comparisons (chaining 2 or more comparison operations) e.g., x > y > z

Compound boolean expression (combine boolean exp using boolean operations and/or etc.) e.g., (1 <= 2) and (2 < 3)

Chained boolean (logical) operations (chaining 2 or more boolean operations) e.g. x and y or z

==========

Break-time Exercise E

True and "Good", False and "Good", True or "Good", False or "Good"
False == False, bool(""), bool("") == False, "" == False

==========

Try the code below:

Q1 True and "Good"

Q2 False and "Good"

Q3 True or "Good"

Q4 False or "Good"

Note: You may get True or False or Good

Also try: "Excellent" and "Good"

Ref: Check "4 and 5" in the Lecture notebook.

Q5 False == False

Q6 bool("")

Q7 bool("") == False

Q8 "" == False

An empty string is regarded as false in a boolean operation (when expect True/False) but
a comparison operation (==) is not a boolean operation, even though it forms a boolean expression.

Interesting: Can compare a string and a number? using ==? using >?

```
print("A"==64) # ok
print("A">64) # not ok
```

==========

Break-time Exercise F

==========

[boolean/logical operators]: And, Or, Not

Using And, Or in daily life. What do the following mean?

(a) If you get average at least 80 and you have taken at least 3 courses, then good job!

*in case average is 80, you have to **continue** ask the total count of courses*

in case average is only 60, no need to ask the total count of courses"

Same thing is done by Python by short-circuit evaluation

(b) if the passenger is sick or i am young and he is old , then I give out my seat.

First: Which of the following will be done?

(1) if the passenger is sick or i am young and he is old , then I give out my seat.

(2) if the passenger is sick or i am young and he is old , then I give out my seat

By comparing **precedence** of or and and, (2) is correct:
conclusion, it is:
if the passenger is sick or i am young and he is old , then I give out my seat

Second: Does **Short-circuit evaluation** apply? How?

in case passenger is sick, no need to check young and old "

Same thing is done by Python "by short-circuit evaluation"

in case passenger is not sick, need to check young and old

A game (in the lecture notebook): input(1) & input(2) & input(3)