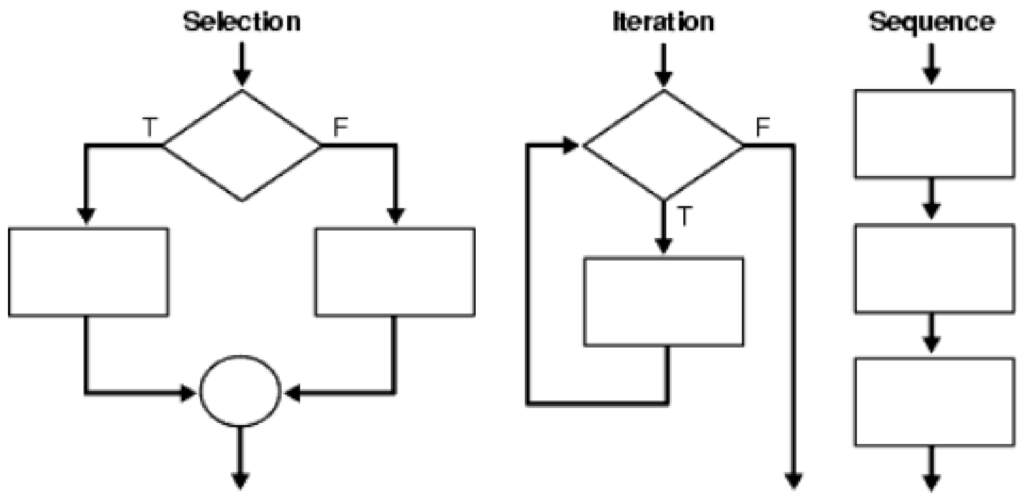
LAB 4: Control structures & Boolean Expressions

A control structure is a statement that controls the order of execution of instructions. There are three fundamental forms of control structures in programming:

Sequential
Selection
Iteration



At the heart of a control structure is the condition. Once you know how to write a condition, writing control structures like if statements or loops is just a breeze.

The **condition** is at the center of control structures. A condition is a comparison expression that gives you a Boolean value. There are only two Boolean values: True and False.

Note: they must be capitalized, as true and false are not defined, hence not keywords of python; and they must NOT be in quotation marks which would make them string, not Boolean.

Although True and False can be used as condition, they are not terribly useful because True would always be true, False would always be false. What we want rather is a condition that could be true or false depending on the data. For that, we need comparison expressions, in this format: <value> <comparison operator> <value>.

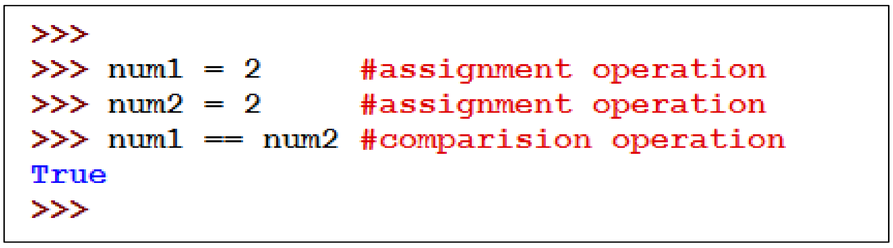
For example: 3 > 5

Comparison operators, and Boolean expressions

Comparison operators compares two values and gives a True or False result. A list of comparison operators are shown below:

Operator	Meaning	Example
==	equal to	age == 4
!=	not equal to	age!=5
<	less than	age < 6
>	greater than	age > 7
<=	less than or equal to	age <=8
>=	greater than or equal to	age >=9

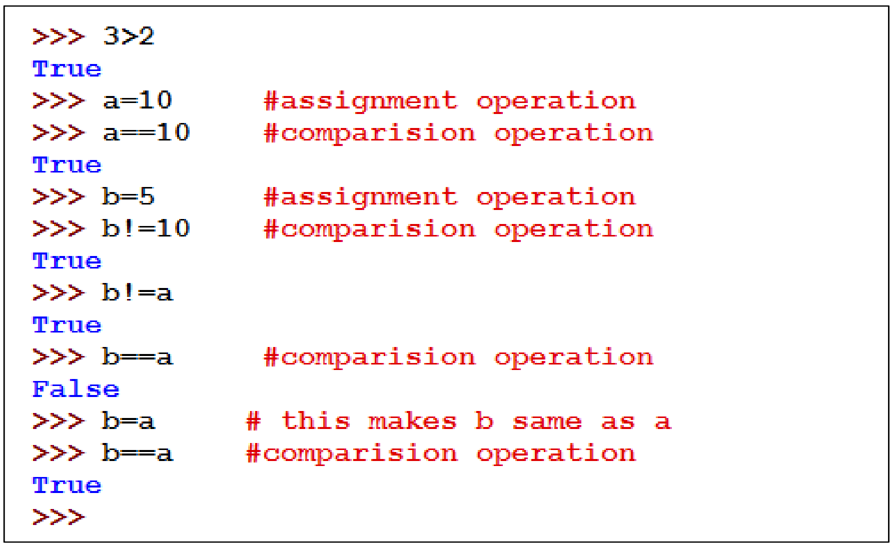
Note: the == operator for determining if two values are equal. Do not confuse this with the assignment operator ☺



```
>>>
>>> num1 = 2      #assignment operation
>>> num2 = 2      #assignment operation
>>> num1 == num2 #comparision operation
True
>>>
```

Orderable values

The order in which values are written makes a difference. Try the following:



```
>>> 3>2
True
>>> a=10      #assignment operation
>>> a==10     #comparision operation
True
>>> b=5       #assignment operation
>>> b!=10     #comparision operation
True
>>> b!=a
True
>>> b==a      #comparision operation
False
>>> b=a      # this makes b same as a
>>> b==a     #comparision operation
True
>>>
```

There is also an order between two alphabetic characters - They normally follow the *dictionary order*.

```
>>> "Alex" == "John"
False
>>> ord("A") # each character has a unicode value
65
>>> ord("B")
66
>>> ord("a")
97
```

There are two 'membership operators' in python which also give True or False: **in** and **not in**. They can be used in similar fashion as the comparison operators.

```
>>> "hello" in "hello world"
True
>>> "morning" not in "hello world"
True
>>> "morning" in "hello world"
False
>>> 10 in [10,20]
True
```

Boolean operators

Quite often, we need to express complex conditions, for example, a condition to see if a number is in the range of 0 and 100. To express such conditions, we have to use multiple comparison expressions.

```
>>> age = 20
>>> age >= 18 and age <=60
True
```

The keyword **and** here is a Boolean operator. There are three Boolean opeators: and, or, not. We can use **and** and **or** to combine comparison expressions. The third boolean operator **not** is used to express the opposite result.

```
>>> age = 20
>>> age >= 18 and age <=60
True
>>> not(age >= 18 and age <=60)   # opposite of above
False
>>>
```

Truth table

Truth table tells us how the combination of multiple comparison expressions are evaluated and in what order.

Truth Table sometimes is also called 'Boolean algebra`, and was developed by George Boole in the mid-1800s. In the truth table, each True or False in the table represents a comparison expression which gives true of false.

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

Checking the **and** operator

```
>>> age = 20
>>> age >= 18 and age <= 60    # True and True
True
>>> age >= 18 and age <= 19    # True and False
False
>>> age >= 25 and age <= 60    # False and True
False
>>> age >= 25 and age <= 19    # False and False
False
```

Checking the **or** operator

```
>>> age >= 18 or age <= 60    # True or True
True
>>> age >= 18 or age <= 19    # True or False
True
>>> age >= 25 or age <= 60    # False or True
True
>>> age >= 25 or age <= 19    # False or False
False
```

Operator precedence

Different operators have different priorities which determine the order the operators are evaluated. This is the operator precedence. If necessary, we can use parentheses to overcome the precedence.

In the order of precedence, Arithmetic is higher than comparison.

Task 1:

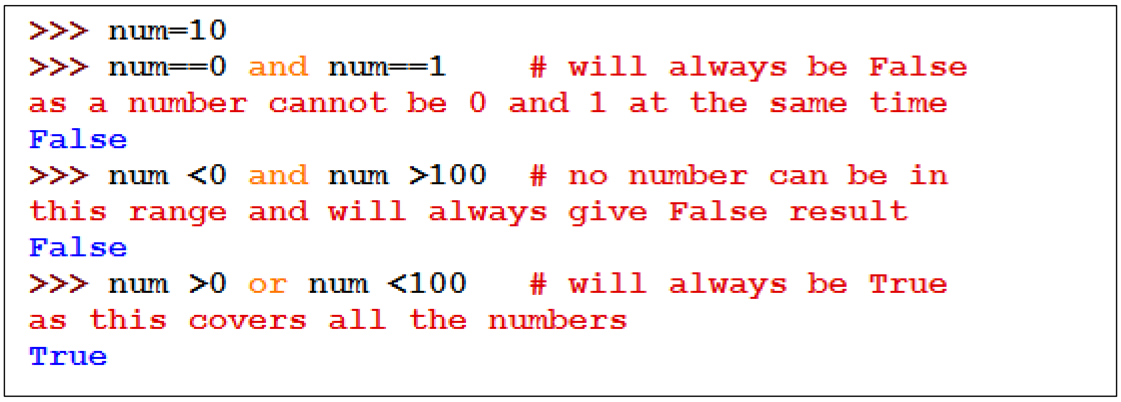
1. write a condition that tests num is greater than or equal to 50
2. write a condition that num is in the range of 50 and 100, inclusive.
3. write a condition for num is greater than or equal to 0, and less than 50
4. write a condition for num is less than 0 or greater than 50
5. write a condition for num is not 0 and not 1

Model answer:

num >= 50
num >= 50 and num <= 100
num >= 0 and num <= 50
num < 0 or num > 50
num != 0 and num != 1

Useless conditions

Some combination of comparison expressions form useless conditions that you should avoid using - they evaluate to the same boolean value no matter what value the participating variable holds:



```
>>> num=10
>>> num==0 and num==1     # will always be False
as a number cannot be 0 and 1 at the same time
False
>>> num <0 and num >100   # no number can be in
this range and will always give False result
False
>>> num >0 or num <100    # will always be True
as this covers all the numbers
True
```

Date Time functions

datetime module

To get the date and time: use the datetime module

```
>>> from datetime import datetime
>>> current_datetime = datetime.now()
>>> print(current_datetime)
2017-02-13 10:30:43.065591
>>>
```

strftime() function

Use the **strftime()** function get a more friendly format (for both date and time)

```
>>> current_datetime.strftime('%Y %B %d %A')
'2017 February 13 Monday'
>>> current_datetime.strftime('%H %M %S')
'10 30 43'
```

For a full list of format codes, check https://docs.python.org/3/library/datetime.html

strptime() function

Use the **strptime()** function to get any date format (from a given date string)

```
>>> dob=input('Please enter your date of birth (dd mm yyyy):')
Please enter your date of birth (dd mm yyyy):1 1 1990
>>>
>>> # Parse the datetime string to a datetime object
>>> dob = datetime.strptime(dob_str, '%d %m %Y')
>>>
>>> # Print in required format
>>> print(dob.strftime('%d %m %Y'))
01 01 1990
>>> print(dob.strftime('%d-%m-%Y'))
01-01-1990
>>> print(dob.strftime('%d/%m/%Y'))
01/01/1990
>>>
>>> # Print smore specific values
>>> print('Your day of birth was a', dob.strftime('%A'))
Your day of birth was a Monday
>>>
```

```
>>> # Calculate the number of days since you were born
>>> num_days= (datetime.now()-dob).days
>>> print('Number of days since your birth is', num_days)
Number of days since your birth is 9905
>>>
```

TASK 2: Write a program **Lab4Task2.py** to ask the user to enter the due date for their assignment. The program should then calculate the number of days till the deadline. Optionally, tell them the answer in the number of weeks plus days.

```
from datetime import datetime

duedate_str = input('Please enter the due date (dd mm yyyy):')

# Parse the datetime string to a datetime object
duedate = datetime.strptime(duedate_str, '%d %m %Y')

# Calculate the number of days to the due date
num_days= (datetime.now()- duedate).days
print('Number of days till due date is', num_days)
```

Sample output

```
>>>
Please enter the due date (dd mm yyyy):1 2 2017
Number of days till due date is 12
>>>
```

Bonus Task:

1. Display the answer in the number of weeks plus days.
   (hint: weeks = num_days // 7 ;days = num_days % 7)

2. If the due date is past (i.e. num_days <0) then give a message that **the project is overdue by ___ days** (do not show days as negative value)

timedelta

TASK 3: Write a program **Lab4Task3.py** to calculate product expiration date at a super market based on manufactured date and product lasting period.

```python
from datetime import datetime, timedelta

mftdate_str = input('Please enter the manufacture date (dd mm yyyy):')
valid_days = int(input('Please enter number of date before expiration: '))

# Parse the datetime string to a datetime object
mftdate = datetime.strptime(mftdate_str, '%d %m %Y')

# Calculate the expiration date
delta_time = timedelta(days = valid_days)
expire_date = mftdate + delta_time

print('Expiry date is', expire_date.strftime('%d-%b-%Y'))
```

Sample output

```
>>>
Please enter the manufacture date (dd mm yyyy):1 1 2017
Please enter number of date before expiration: 30
Expiry date is 31-Jan-2017
>>>
```