

Exposure CS 2021 **for CS1**

Chapter 11 Slides

Boolean Logic

**PowerPoint Presentation
created by:
Mr. John L. M. Schram
and Mr. Leon Schram
Authors of Exposure
Computer Science**



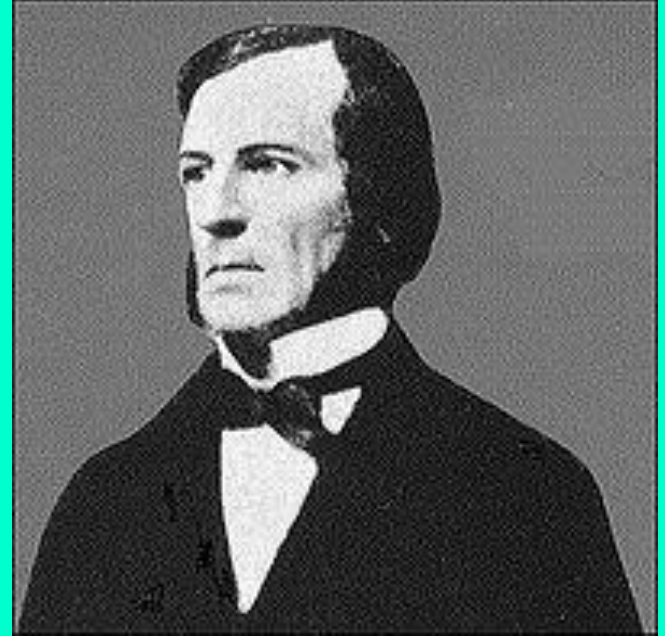
Section 11.1

Introduction

George Boole

More than 160 years ago, there was a mathematician, *George Boole*, who founded a branch of mathematics called *Boolean Algebra*.

In *Boolean Algebra*, everything is either **True** or **False**. Statements that are either **True** or **False** are called *Boolean Statements*.



The conditions you used with *selection* and *repetition* back in Chapters 7 & 8 were all *Boolean statements*.

Python has a **bool** datatype named after George Boole.

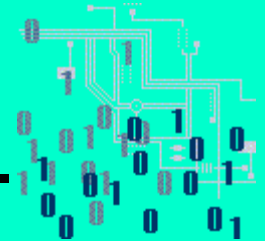
Section 11.2

**What is a
Boolean
statement?**

What is a Boolean Statement?

The sentence, statement, condition, whatever, must be **True** or **False**.

No questions, no ambiguities, no arguments.



The basis of processing data is the binary system of *on* and *off*, or *1* and *0*, which certainly sounds a lot like **True** or **False**.

Each of the following five statements is a Boolean statement:

A mile is longer than a kilometer.

July and August both have the same number of days.

A pound of feathers is lighter than a pound of lead.

The Moon is larger than the Sun.

New York City has more people than Baltimore.

Boolean Statement Examples

English Sentence	Boolean Statement	T/F
<i>A mile is longer than a kilometer.</i>	<code>Mile > Kilometer</code>	True
<i>July and August have the same days.</i>	<code>JulDays == AugDays</code>	True
<i>A pound of feathers is lighter than a pound of lead.</i>	<code>PoundF < PoundL</code>	False
<i>The Moon is larger than the Sun.</i>	<code>MoonSize > SunSize</code>	False
<i>New York City has more people than Baltimore.</i>	<code>NYPop > BaltPop</code>	True

Sentences are not always so short and straight forward. Frequently there are multiple conditions in one statement. Special rules need to be followed to determine if the entire statement is **True** or **False**.

Consider the following sentences with compound conditions:

She is a computer science teacher and she is a math teacher.

The number is odd or the number is even.

Enter again if gender is not male or gender is not female.

Employment requires a CPA and five years experience.

The same sentences converted into Boolean statements are:

`She == CSTeacher and She == MathTeacher`

`Number % 2 == 1 or Number % 2 == 0`

`Gender != 'M' or Gender != 'F'`

`CPA == 'Y' and experience >= 5`

Section 11.3

Boolean Operators

Logical OR Example

You are admitted to a particular college if your SAT scores are above 1100 OR you are in the top 20% of your graduating class. This chart shows her 4 possible cases:



SAT >1100?	Top 20%?	Admitted?
YES	YES	
YES	NO	
NO	YES	
NO	NO	

Logical OR Example

You are admitted to a particular college if your SAT scores are above 1100 OR you are in the top 20% of your graduating class. This chart shows her 4 possible cases:



SAT >1100?	Top 20%?	Admitted?
YES	YES	Yes
YES	NO	Yes
NO	YES	Yes
NO	NO	No

Boolean Operators

Boolean OR

A	B	A or B
T	T	T
T	F	T
F	T	T
F	F	F

Logical AND Example

Your parents tell you that you can go to the movies if you finish your homework **AND** clean your room. This chart shows 4 possible cases:



Homework?	Clean Room?	Go to Movies?
YES	YES	
YES	NO	
NO	YES	
NO	NO	

Logical AND Example

Your parents tell you that you can go to the movies if you finish your homework **AND** clean your room. This chart shows 4 possible cases:



Homework?	Clean Room?	Go to Movies?
YES	YES	Yes
YES	NO	No
NO	YES	No
NO	NO	No

Boolean Operators

Boolean AND

A	B	A and B
T	T	T
T	F	F
F	T	F
F	F	F

Boolean Operators

Boolean NOT

A
T
F

not A
F
T

Section 11.4

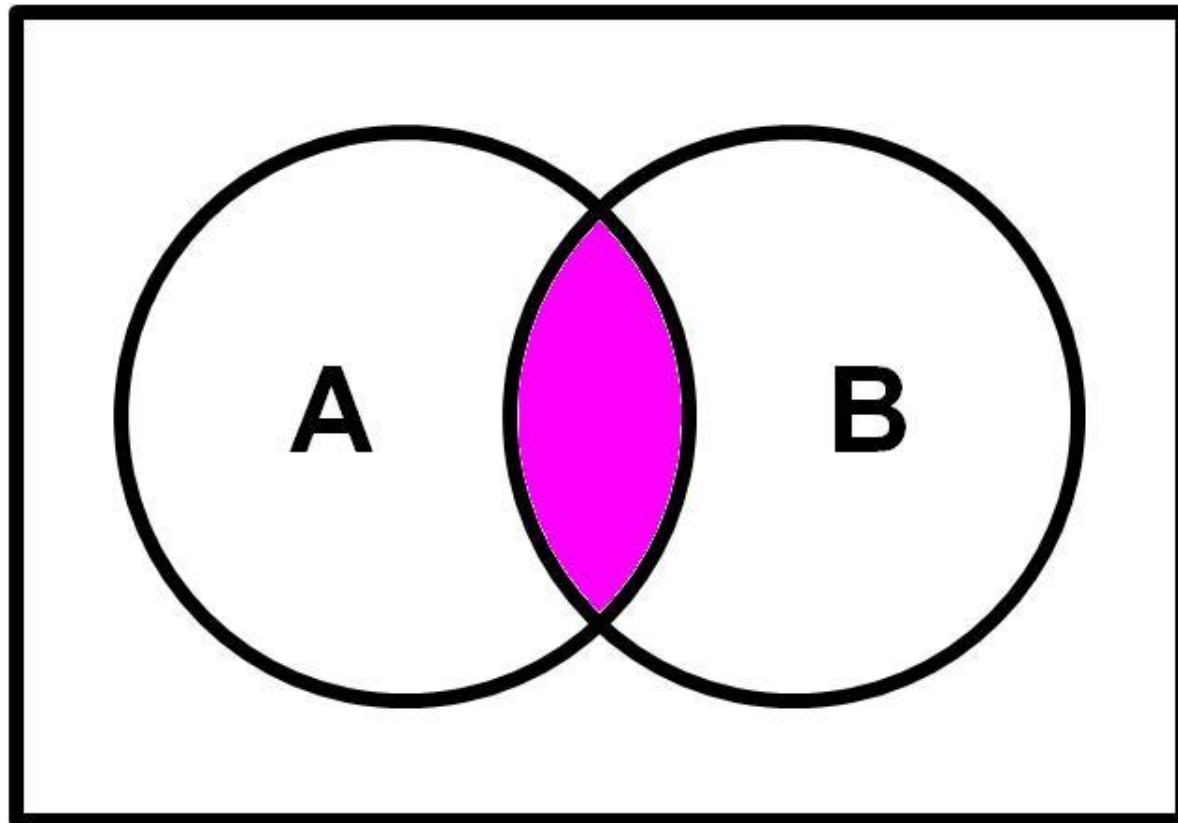
Boolean Algebra

& Venn Diagrams

Venn Diagram #1

The Boolean Algebra logical **and** ($*$) can be demonstrated with Venn Diagrams, using *intersection*.

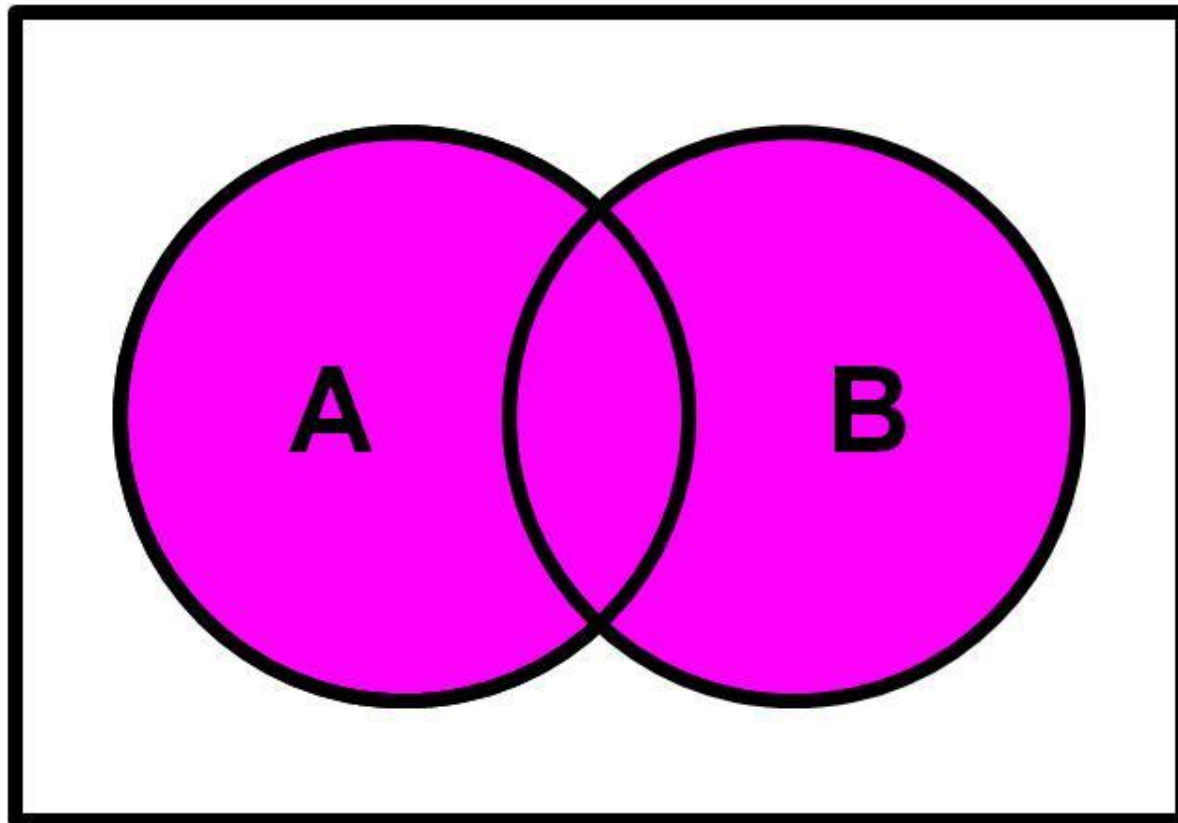
A intersect B also A and B also $A * B$ also AB



Venn Diagram #2

The Boolean Algebra logical **or** ($+$) can be demonstrated with Venn Diagrams, using *union*.

A union B also **A or B** also **A + B**

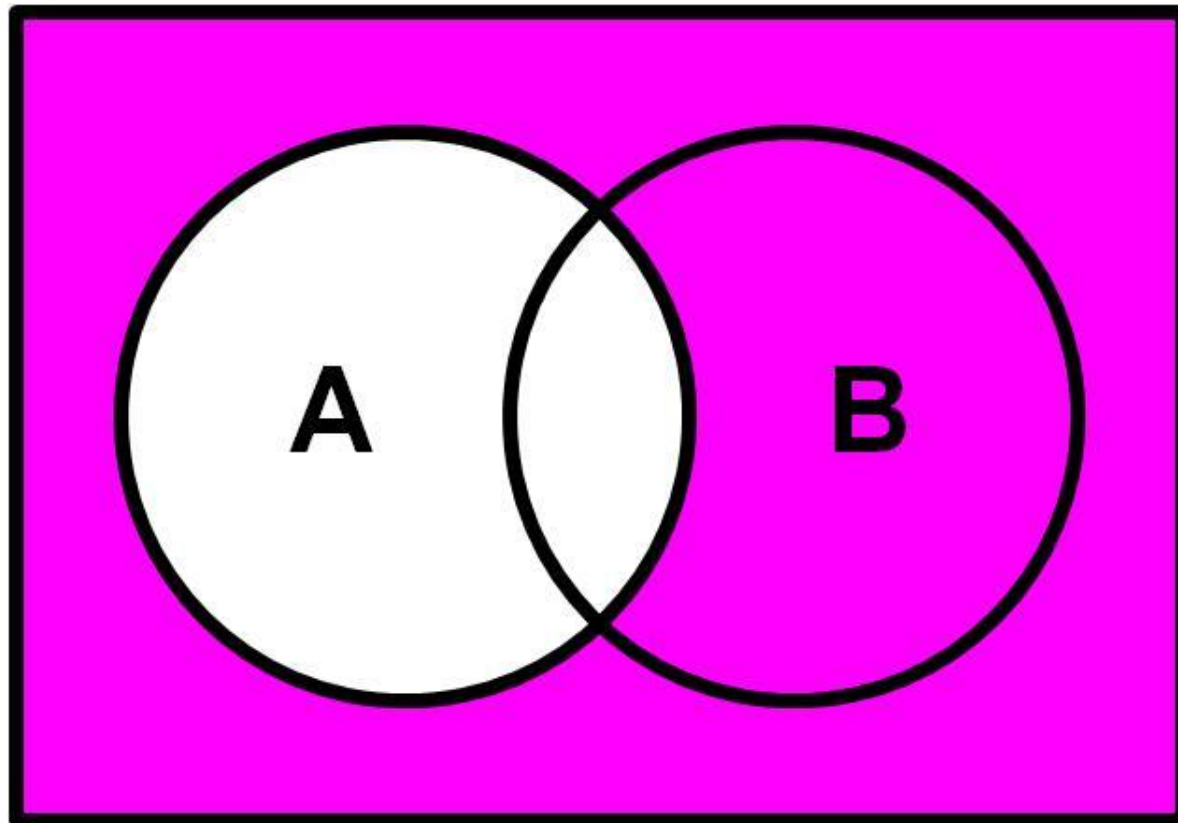


Venn Diagram #3

The Boolean Algebra logical **not**(\sim)

not A also $\sim A$

This is the negation or the opposite of A.

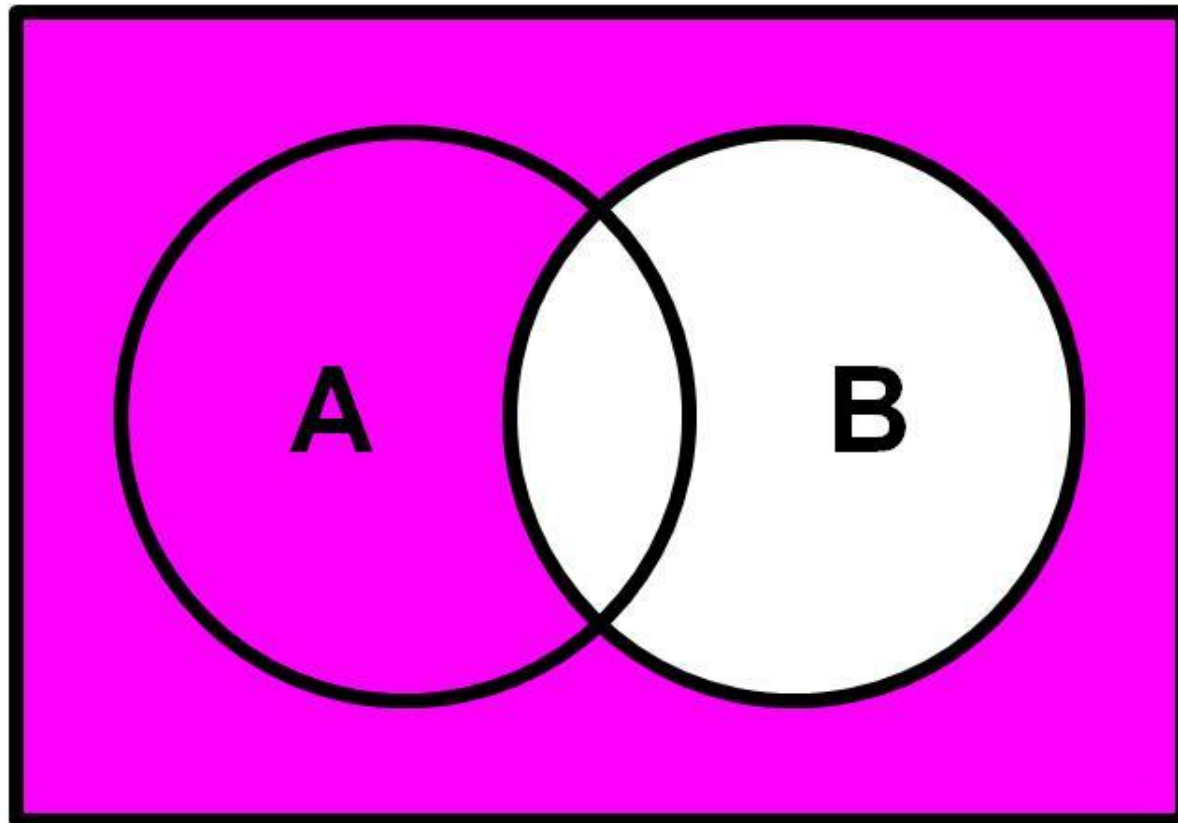


Venn Diagram #4

The Boolean Algebra logical **not**(\sim)

not B also $\sim B$

This is the negation or the opposite of B.

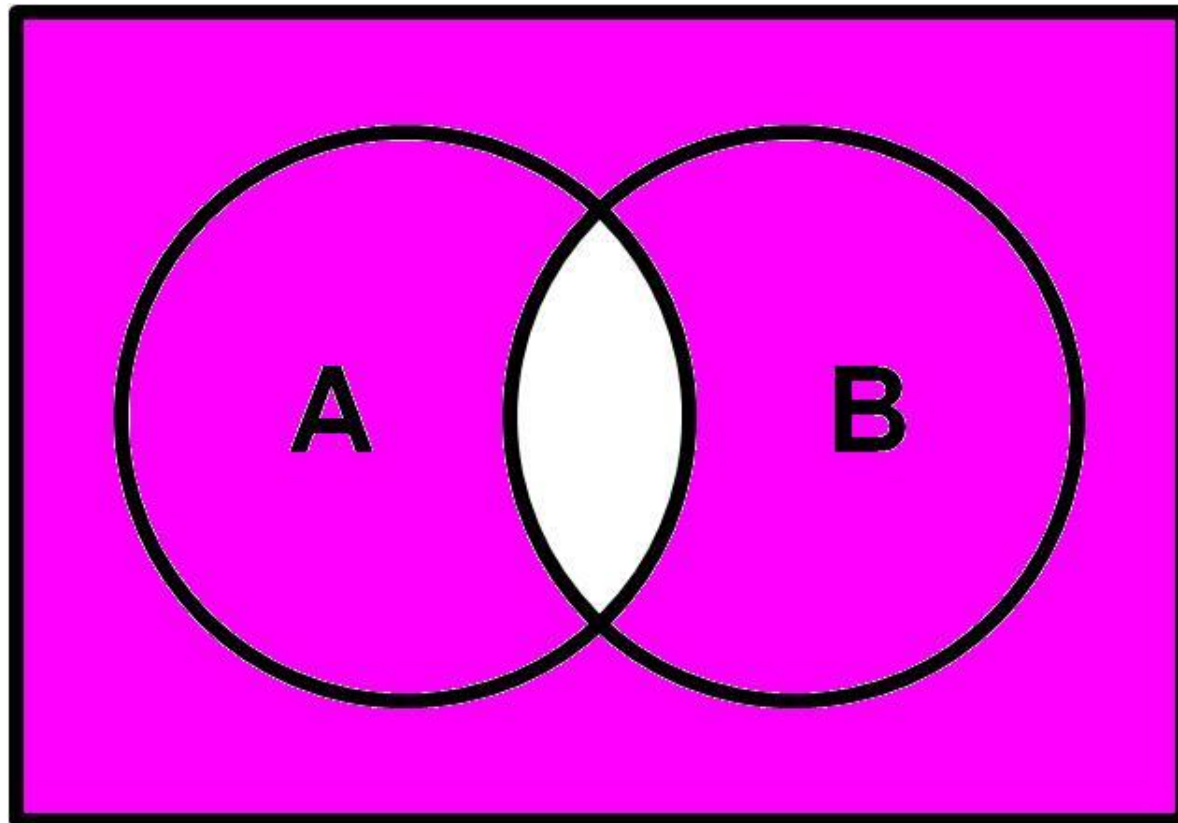


Venn Diagram #5

not (A and B)

$\sim(A * B)$

This is the opposite of (A and B).

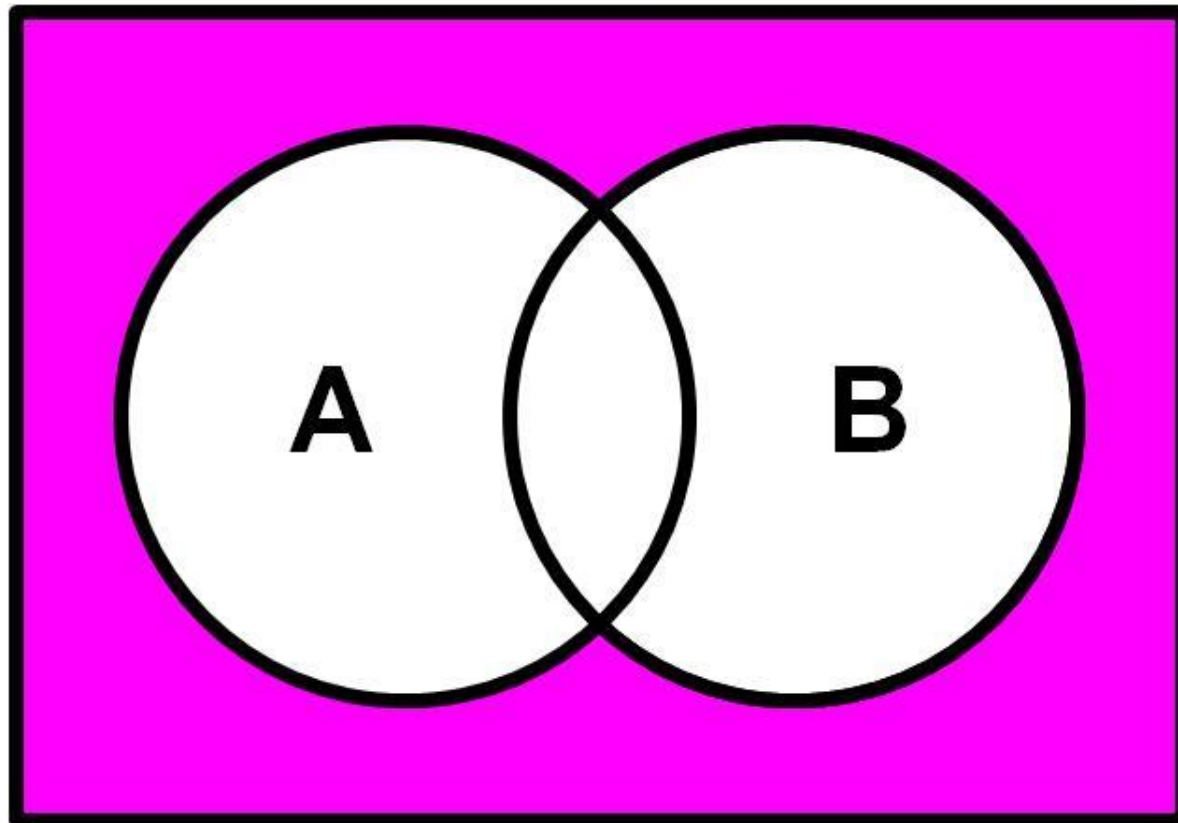


Venn Diagram #6

not (A or B)

$\sim(A + B)$

This is the opposite of (A or B).

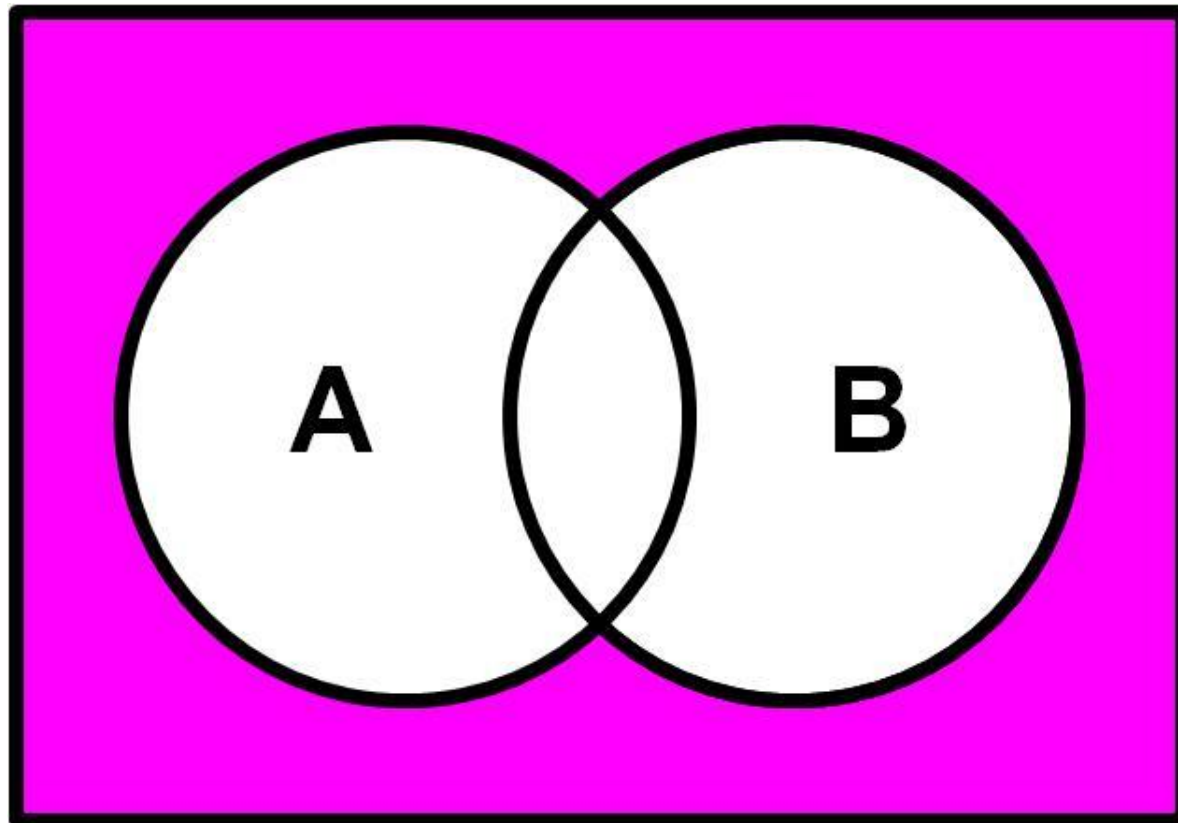


Venn Diagram #7

not A and not B

$$\sim A * \sim B$$

This is identical to not(A or B).

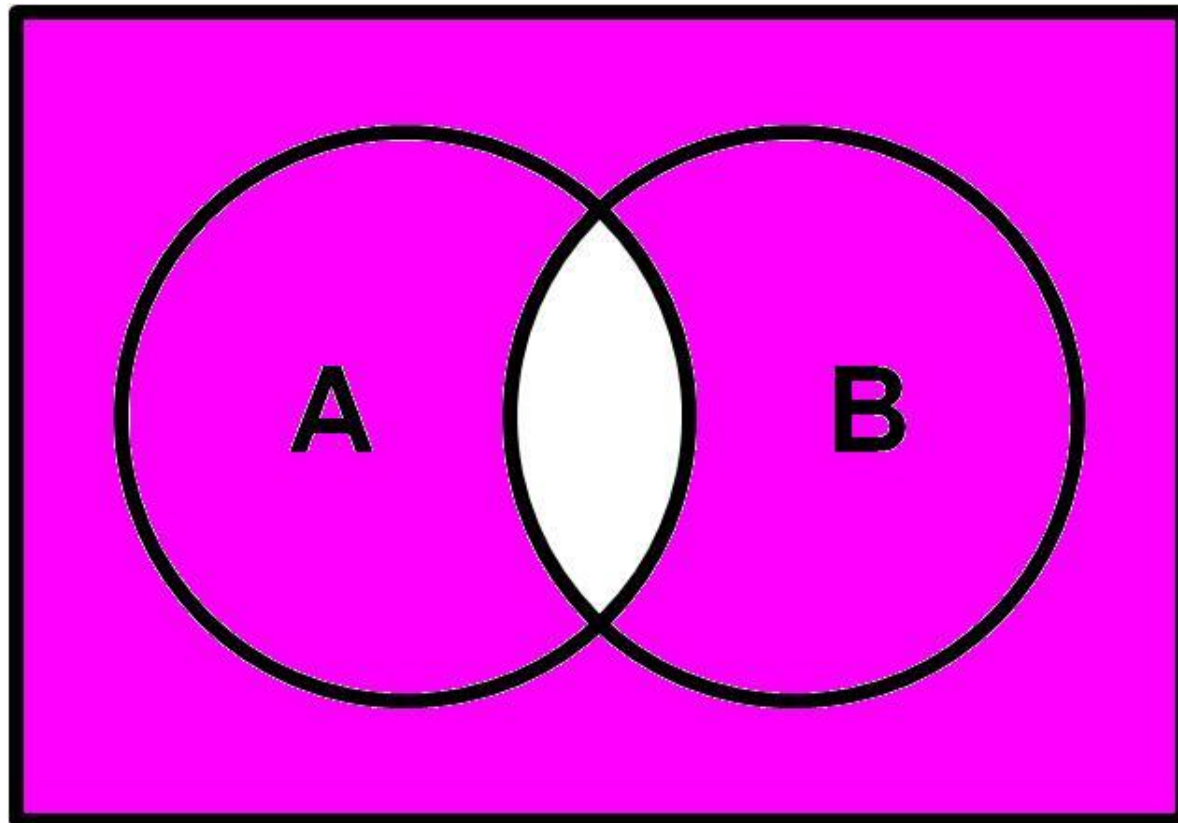


Venn Diagram #8

not A or not B

$$\sim A + \sim B$$

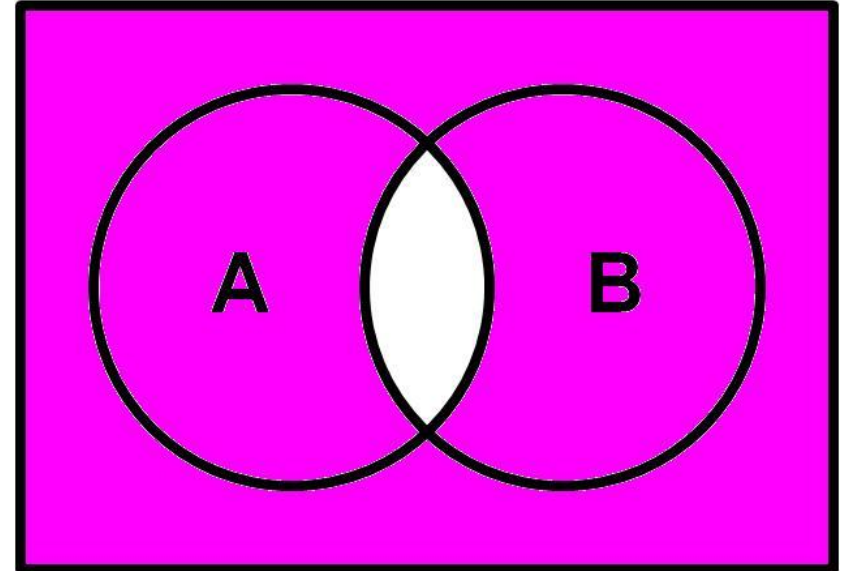
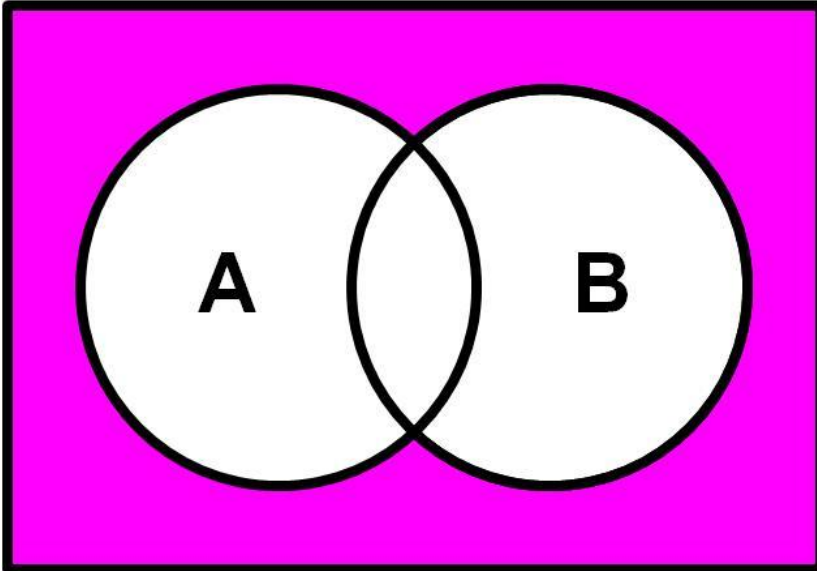
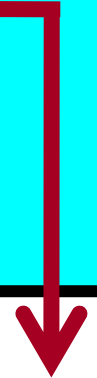
This is identical to not(A and B).



DeMorgan's Law

$\text{not}(A \text{ and } B) = \text{not } A \text{ or not } B$

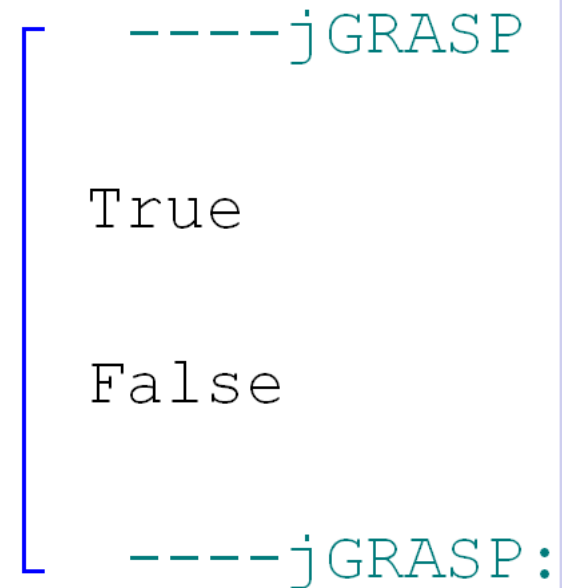
$\text{not}(A \text{ or } B) = \text{not } A \text{ and not } B$



Section 11.5

Boolean Values and Variables

```
1 # Boolean01.py
2 # This program demonstrates a Boolean expression
3 # being used in an <if> statement.
4
5
6
7 x = 10
8
9 print()
10 if x == 10:
11     print("True")
12 else:
13     print("False")
14
15 print()
16 if x == 5:
17     print("True")
18 else:
19     print("False")
```



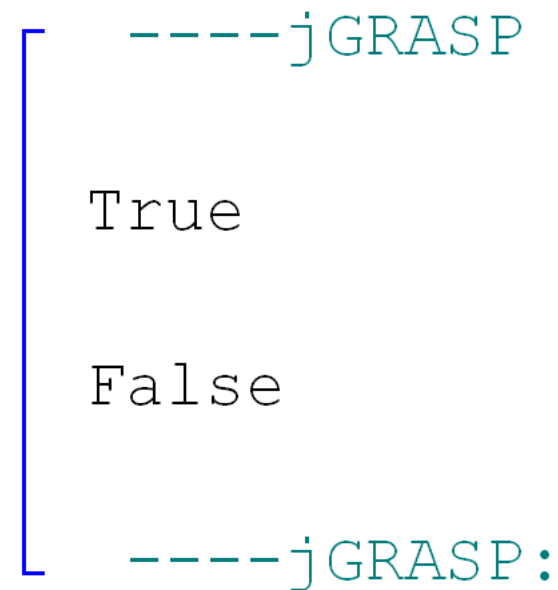
----j GRASP

True

False

----j GRASP:

```
1 # Boolean02.py
2 # This program demonstrates that conditional
3 # statements have a Boolean value which is
4 # either <True> or <False> and these values
5 # can be displayed.
6
7
8 x = 10
9
10 print()
11 print(x == 10)
12 print()
13 print(x == 5)
14
```



```
----jGRASP
True
False
----jGRASP:
```

Output for the next 3 programs...

```
----jGRASP exec: python Boolean03.py
```

▶▶ What is the GCF of 120 and 108? --> 1

▶▶ What is the GCF of 120 and 108? --> 2

▶▶ What is the GCF of 120 and 108? --> 3

▶▶ What is the GCF of 120 and 108? --> 12

You answered it correctly after 4 attempts.

```
----jGRASP: operation complete.
```

```
1 # Boolean03.py
2 # This program shows a Boolean expression
3 # being used in a <while> loop.
4
5
6 gcf = 0
7 attempt = 0
8
9 while gcf != 12:
10     attempt += 1
11     gcf = eval(input("\nWhat is the GCF of 120 and 108? --> "))
12
13 if attempt == 1:
14     print("\nYou got it on the first try!")
15 else:
16     print("\nYou answered it correctly after",attempt,"attempts.")
17
```

**This program
will repeat
until a correct
GCF of 12
is entered.**

```
1 # Boolean04.py
2 # This program demonstrates a practical use of the <bool>
3 # data type, which only has two values, <True> and <False>.
4 # NOTE: Boolean variables add readability to programs.
5
6
7 gcf = 0
8 attempt = 0
9 correct = False
10
11 while not correct:
12     attempt += 1
13     gcf = eval(input("\nWhat is the GCF of 120 and 108?  -->  "))
14     if gcf == 12:
15         correct = True
16     else:
17         correct = False
18
19 if attempt == 1:
20     print("\nYou got it on the first try!")
21 else:
22     print("\nYou answered it correctly after",attempt,"attempts.")
```

```
1 # Boolean05.py
2 # This program executes in the same manner as Boolean04.py.
3 # The abbreviated Boolean assignment statement is used in
4 # place of the longer <if...else> expression.
5
6
7
8 gcf = 0
9 attempt = 0
10 correct = False
11
12 while not correct:
13     attempt += 1
14     gcf = eval(input("\nWhat is the GCF of 120 and 108?  --> "))
15     correct = gcf == 12
16
17 if attempt == 1:
18     print("\nYou got it on the first try!")
19 else:
20     print("\nYou answered it correctly after",attempt,"attempts.")
21
```



```
1 # Boolean06.py
2 # This program assigns the values of 4 different
3 # expressions to 4 different variables.
4 # The intent is to show that storing the values
5 # of Boolean expressions is no different than
6 # storing the values of any other expressions.
7
8
9 x = 23 + 45                    # Integer expression
10
11 y = 6.33 * 6.31 * 6.23       # Real Number expression
12
13 name = "John " + "Smith"     # String expression
14
15 average = 85
16 passing = average >= 70      # Boolean expression
17
18 print()
19 print("Integer value:        ",x)
20 print("Real Number value:    ",y)
21 print("String value:         ",name)
22 print("Boolean value:        ",passing)
```

```

1 # Boolean06.
2 # This progr
3 # expressior
4 # The intent
5 # of Boolean
6 # storing th
7
8
9 x = 23 + 45
10
11 y = 6.33 * 6.31 * 6.23           # Real Number expression
12
13 name = "John " + "Smith"         # String expression
14
15 average = 85
16 passing = average >= 70          # Boolean expression
17
18 print()
19 print("Integer value:           ",x)
20 print("Real Number value:",y)
21 print("String value:           ",name)
22 print("Boolean value:          ",passing)

```

```

----jGRASP exec: python Boolean06

Integer value:           68
Real Number value:      248.840529
String value:           John Smith
Boolean value:           True

----jGRASP: operation complete.

```

Section 11.6

compound
conditions

```
1 # CompoundCondition01.py
2 # This program demonstrates compound decisions
3 # with the logical <or> operator.
4
5
6 print()
7 education = eval(input("Enter years of education --> "))
8 experience = eval(input("Enter years of experience --> "))
9 print()
10
11 if education >= 16 or experience >= 5:
12     print("You are hired!")
13 else:
14     print("You are not qualified.")
15
```

The “Nice Boss” Using OR Hires you if at least one condition is True.

```
----jGRASP exec: python CompoundCondition01.py
>> Enter years of education --> 16
>> Enter years of experience --> 0
You are hired!
----jGRASP: operation complete.

----jGRASP exec: python CompoundCondition01.py
>> Enter years of education --> 13
>> Enter years of experience --> 7
You are hired!
----jGRASP: operation complete.

----jGRASP exec: python CompoundCondition01.py
>> Enter years of education --> 18
>> Enter years of experience --> 10
You are hired!
----jGRASP: operation complete.

----jGRASP exec: python CompoundCondition01.py
>> Enter years of education --> 12
>> Enter years of experience --> 3
You are not qualified.
----jGRASP: operation complete.
```

```
1 # CompoundCondition02.py
2 # This program demonstrates compound decisions
3 # with the logical <and> operator.
4
5
6 print()
7 education = eval(input("Enter years of education --> "))
8 experience = eval(input("Enter years of experience --> "))
9 print()
10
11 if education >= 16 and experience >= 5:
12     print("You are hired!")
13 else:
14     print("You are not qualified.")
15
```

The “Picky Boss”

Using AND

Hires you
only if both
conditions
are True.

```
----jGRASP exec: python CompoundCondition02.py
>> Enter years of education --> 16
>> Enter years of experience --> 0

You are not qualified.

----jGRASP: operation complete.

----jGRASP exec: python CompoundCondition02.py
>> Enter years of education --> 13
>> Enter years of experience --> 7

You are not qualified.

----jGRASP: operation complete.

----jGRASP exec: python CompoundCondition02.py
>> Enter years of education --> 18
>> Enter years of experience --> 10

You are hired!

----jGRASP: operation complete.

----jGRASP exec: python CompoundCondition02.py
>> Enter years of education --> 12
>> Enter years of experience --> 3

You are not qualified.

----jGRASP: operation complete.
```

```
1 # CompoundCondition03.py
2 # This program demonstrates the logical <not> operator.
3 # NOTE: This also requires (parentheses).
4
5
6 print()
7 education = eval(input("Enter years of education --> "))
8 experience = eval(input("Enter years of experience --> "))
9 print()
10
11 if not (education >= 16 or experience >= 5):
12     print("You are hired!")
13 else:
14     print("You are not qualified.")
15
```


The
“Crazy Boss”

Using NOT

Hires you
only if both
conditions
are False.

```
----jGRASP exec: python CompoundCondition03.py
>> Enter years of education --> 16
>> Enter years of experience --> 0

You are not qualified.

----jGRASP: operation complete.

----jGRASP exec: python CompoundCondition03.py
>> Enter years of education --> 13
>> Enter years of experience --> 7

You are not qualified.

----jGRASP: operation complete.

----jGRASP exec: python CompoundCondition03.py
>> Enter years of education --> 18
>> Enter years of experience --> 10

You are not qualified.

----jGRASP: operation complete.

----jGRASP exec: python CompoundCondition03.py
>> Enter years of education --> 12
>> Enter years of experience --> 3

You are hired!

----jGRASP: operation complete.
```

Section 11.7

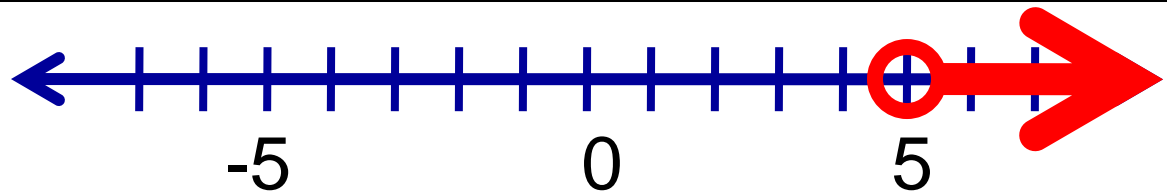
Ranges

Quick Algebra Review

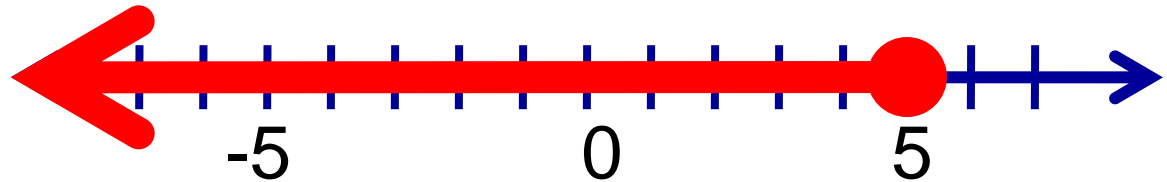
Number Lines

Consider these number lines:

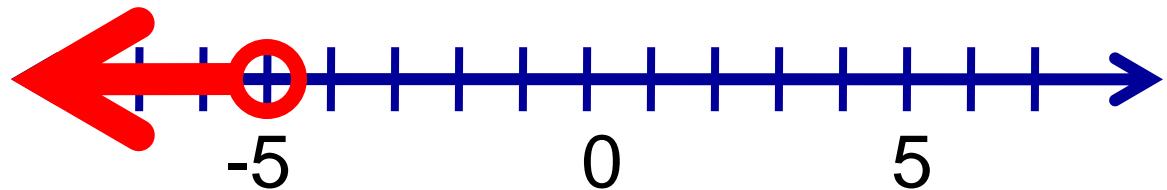
$$x > 5$$



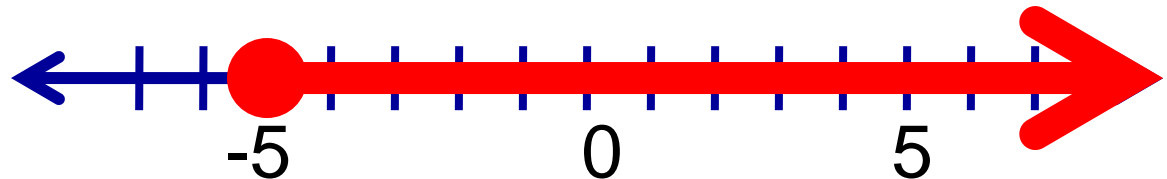
$$x \leq 5$$



$$x < -5$$



$$x \geq -5$$

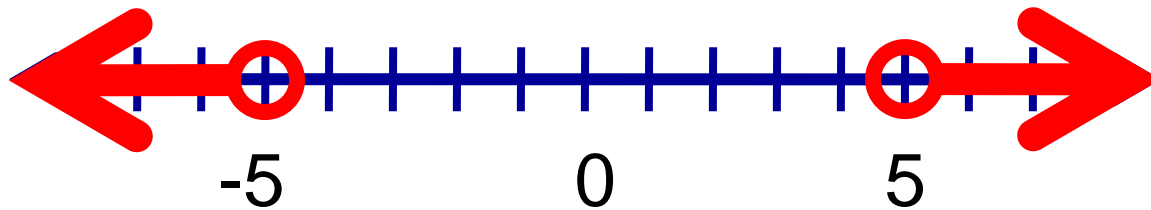


Quick Algebra Review

2 Inequalities with OR

Consider this compound inequality:

$$x < -5 \text{ OR } x > 5$$



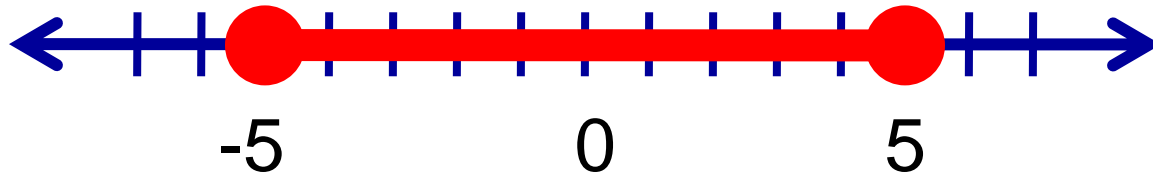
This is an **Outside Range** and it represents all numbers that are NOT between -5 and 5.

Quick Algebra Review

2 Inequalities with AND

Consider this compound inequality:

$$x \geq -5 \text{ AND } x \leq 5$$



This is an **Inside Range** and it represents all numbers that ARE between -5 and 5.

```
1 # Ranges01.py
2 # This program determines if an age is between 16 and 90.
3 # The range is defined using a compound condition with <and>.
4 # NOTE: This range logic works in any programming language.
5
6
7 print()
8 age = eval(input("How old are you? --> "))
9 print()
10
11 if age >= 16 and age <= 90:
12     print("You are the proper age to drive.")
13 else:
14     print("You are not the proper age to drive.")
15
```

Valid Age Range: 16 – 90



```
----jGRASP exec: python Ranges01.py
> How old are you? --> 16
You are the proper age to drive.
----jGRASP: operation complete.

----jGRASP exec: python Ranges01.py
> How old are you? --> 50
You are the proper age to drive.
----jGRASP: operation complete.

----jGRASP exec: python Ranges01.py
> How old are you? --> 13
You are not the proper age to drive.
----jGRASP: operation complete.

----jGRASP exec: python Ranges01.py
> How old are you? --> 107
You are not the proper age to drive.
----jGRASP: operation complete.
```

```
1 # Ranges02.py
2 # This program demonstrates a shortcut way to handle ranges.
3 # NOTE: This range shortcut only works in Python.
4
5
6 print()
7 age = eval(input("How old are you? --> "))
8 print()
9
10 if 16 <= age <= 90:
11     print("You are the proper age to drive.")
12 else:
13     print("You are not the proper age to drive.")
14
```



```
1 # Ranges03.py
2 # This program defines several different ranges
3 # using an <if..elif..else> structure.
4
5
6 print()
7 gpa = eval(input("What is your GPA? --> "))
8 print()
9
10 if gpa >= 3.9:
11     print("Summa Cum Laude")
12 elif gpa >= 3.75:
13     print("Magna Cum Laude")
14 elif gpa >= 3.5:
15     print("Cum Laude")
16 elif gpa >= 2.65:
17     print("Graduate without Honors")
18 else:
19     print("Did not graduate")
```



```
----jGRASP exec: python Ranges03.py
>> What is your GPA? --> 4.0
Summa Cum Laude
----jGRASP: operation complete.

----jGRASP exec: python Ranges03.py
>> What is your GPA? --> 3.8
Magna Cum Laude
----jGRASP: operation complete.

----jGRASP exec: python Ranges03.py
>> What is your GPA? --> 3.5
Cum Laude
----jGRASP: operation complete.

----jGRASP exec: python Ranges03.py
>> What is your GPA? --> 2.8
Graduate without Honors
----jGRASP: operation complete.

----jGRASP exec: python Ranges03.py
>> What is your GPA? --> 2.3
Did not graduate
----jGRASP: operation complete.
```

Section 11.8

Input



Protection

```
1 # InputProtection01.py
2 # This program enters the SAT score, gender and last name
3 # of a college applicant.  Nothing prevents the user from
4 # entering invalid information for <sat> and <gender>.
5
6
7 print()
8 sat = eval(input("Enter SAT {400..1600} --> "))
9 gender = input("Enter your gender {M/F} --> ")
10 lastName = input("Enter your last name --> ")
11 print()
12
13 if gender == 'M' or gender == 'm':
14     print("Mr.",lastName,end = ", ")
15 if gender == 'F' or gender == 'f':
16     print("Ms.",lastName,end = ", ")
17
18 if sat >= 1100:
19     print("you are admitted!")
20 else:
21     print("you are not admitted.")
22
```

**Valid SAT
Range:
400 – 1600**

**Valid Gender:
'M', 'm', 'F', 'f'**

```
----jGRASP exec: python InputProtection01.py
>>> Enter SAT {400..1600} --> 1200
>>> Enter your gender {M/F} --> F
>>> Enter your last name --> Jones

Ms. Jones, you are admitted!

----jGRASP: operation complete.

----jGRASP exec: python InputProtection01.py
>>> Enter SAT {400..1600} --> 900
>>> Enter your gender {M/F} --> M
>>> Enter your last name --> Smith

Mr. Smith, you are not admitted.

----jGRASP: operation complete.

----jGRASP exec: python InputProtection01.py
>>> Enter SAT {400..1600} --> 5000000
>>> Enter your gender {M/F} --> m
>>> Enter your last name --> Jackson

Mr. Jackson, you are admitted!

----jGRASP: operation complete.

----jGRASP exec: python InputProtection01.py
>>> Enter SAT {400..1600} --> -100
>>> Enter your gender {M/F} --> z
>>> Enter your last name --> Green

you are not admitted.

----jGRASP: operation complete.
```

```
1 # InputProtection02.py
2 # This program improves on the previous program by
3 # adding 2 <while> loops that will force the user to
4 # keep re-entering the information until it is valid.
5
6
7 print()
8
9 sat = 0
10 while not(sat >= 400 and sat <= 1600):
11     sat = eval(input("Enter SAT {400..1600} --> "))
12
13 gender = ''
14 while not(gender == 'M' or gender == 'F'):
15     gender = input("Enter your gender {M/F} --> ")
16
17 lastName = input("Enter your last name --> ")
18 print()
19
20 if gender == 'M':
21     print("Mr.",lastName,end = ", ")
22 if gender == 'F':
23     print("Ms.",lastName,end = ", ")
24
25 if sat >= 1100:
26     print("you are admitted!")
27 else:
28     print("you are not admitted.")
```

```
----jGRASP exec: python InputProtection02
```

```
▶▶ Enter SAT {400..1600} --> 100
▶▶ Enter SAT {400..1600} --> 2000
▶▶ Enter SAT {400..1600} --> 1300
▶▶ Enter your gender {M/F} --> Z
▶▶ Enter your gender {M/F} --> 1
▶▶ Enter your gender {M/F} --> #
▶▶ Enter your gender {M/F} --> M
▶▶ Enter your last name --> Smith
```

Mr. Smith, you are admitted!

```
----jGRASP: operation complete.
```

```
1 # InputProtection03.py
2 # This program "distributes the not" in the
3 # <while> loops using DeMorgan's Law.
4
5
6 print()
7
8 sat = 0
9 while sat < 400 or sat > 1600:
10     sat = eval(input("Enter SAT {400..1600} --> "))
11
12 gender = ''
13 while gender != 'M' and gender != 'F':
14     gender = input("Enter your gender {M/F} --> ")
15
16 lastName = input("Enter your last name --> ")
17 print()
18
19 if gender == 'M':
20     print("Mr.",lastName,end = ", ")
21 if gender == 'F':
22     print("Ms.",lastName,end = ", ")
23
24 if sat >= 1100:
25     print("you are admitted!")
26 else:
27     print("you are not admitted.")
```



```
1 # InputProtection04.py
2 # This program uses Boolean variables to make the
3 # program more readable.  Addition <if> statements
4 # are also added to make the program more user-friendly.
5
6
7 sat = 0
8 satOK = False
9 while not satOK:
10     sat = eval(input("\nEnter SAT {400..1600} --> "))
11     satOK = 400 <= sat <= 1600
12     if not satOK:
13         print("\nError! Please enter a number between 400 & 1600.")
14
15 gender = ''
16 genderOK = False
17 while not genderOK:
18     gender = input("\nEnter your gender {M/F} --> ")
19     genderOK = gender in ['M', 'm', 'F', 'f'] # another Python shortcut
20     if not genderOK:
21         print("\nError! Please enter either an 'M' or an 'F'.")
22
23 lastName = input("\nEnter your last name --> ")
24 print()
25
26 if gender == 'M' or gender == 'm':
27     print("Mr.", lastName, end = ", ")
28 if gender == 'F' or gender == 'f':
29     print("Ms.", lastName, end = ", ")
30
31 if sat >= 1100:
32     print("you are admitted!")
33 else:
34     print("you are not admitted.")
```

**Valid SAT
Range:
400 – 1600**

**Valid Gender:
'M', 'm', 'F', 'f'**

```
----jGRASP exec: python InputProtection04.py
> Enter SAT {400..1600} --> 100
Error! Please enter a number between 400 & 1600.
> Enter SAT {400..1600} --> 2000
Error! Please enter a number between 400 & 1600.
> Enter SAT {400..1600} --> -100
Error! Please enter a number between 400 & 1600.
> Enter SAT {400..1600} --> 2000000000
Error! Please enter a number between 400 & 1600.
> Enter SAT {400..1600} --> 900
> Enter your gender {M/F} --> W
Error! Please enter either an 'M' or an 'F'.
> Enter your gender {M/F} --> 7
Error! Please enter either an 'M' or an 'F'.
> Enter your gender {M/F} --> @
Error! Please enter either an 'M' or an 'F'.
> Enter your gender {M/F} --> F
> Enter your last name --> Jones
Ms. Jones, you are not admitted.
----jGRASP: operation complete.
```

Section 11.9

Logic Errors



```
1 # LogicError01.py
2 # This program is similar to program InputProtection02.py,
3 # but now there are 2 logic errors because the parentheses
4 # in the <while> statements do not contain the entire Boolean
5 # expression. This means the <not> only applies to the first
6 # part of the expression. When you run the program it may
7 # seem to work, unless you are female or enter an SAT score
8 # above 1600.
9
10
11 print()
12
13 sat = 0
14 while not(sat >= 400) and sat <= 1600:
15     sat = eval(input("Enter SAT {400..1600} --> "))
16
17 gender = ''
18 while not(gender == 'M') or gender == 'F':
19     gender = input("Enter your gender {M/F} --> ")
20
21 lastName = input("Enter your last name --> ")
22 print()
23
24 if gender == 'M':
25     print("Mr.",lastName,end = ", ")
26 if gender == 'F':
27     print("Ms.",lastName,end = ", ")
28
29 if sat >= 1100:
30     print("you are admitted!")
31 else:
32     print("you are not admitted.")
```

```
-----jGRASP exec: python LogicError01
```

```

>> Enter SAT {400..1600} --> -100
>> Enter SAT {400..1600} --> 2000
>> Enter your gender {M/F} --> #
>> Enter your gender {M/F} --> F
>> Enter your gender {M/F} --> M
>> Enter your last name --> Smith
```

```
Mr. Smith, you are admitted!
```

```
-----jGRASP: operation complete.
```

```
1 # LogicError02.py
2 # This program is similar to program InputProtection03.py,
3 # but now there is a subtle logic error in the first loop.
4 # The opposite is sat >= 400 is sat < 400, NOT sat <= 400.
5 # The opposite is sat <= 1600 is sat > 1600, NOT sat >= 1600.
6 # While the program works most of the time, it will not work
7 # for the "border cases" when the SAT is exactly 400 or 1600.
8
9
10 print()
11
12 sat = 0
13 while sat <= 400 or sat >= 1600:
14     sat = eval(input("Enter SAT {400..1600} --> "))
15
16 gender = ''
17 while gender != 'M' and gender != 'F':
18     gender = input("Enter your gender {M/F} --> ")
19
20 lastName = input("Enter your last name --> ")
21 print()
22
23 if gender == 'M':
24     print("Mr.", lastName, end = ", ")
25 if gender == 'F':
26     print("Ms.", lastName, end = ", ")
27
28 if sat >= 1100:
29     print("you are admitted!")
30 else:
31     print("you are not admitted.")
```

```
----jGRASP exec: python LogicError02
```

```
▶▶ Enter SAT {400..1600} --> 100
▶▶ Enter SAT {400..1600} --> 2000
▶▶ Enter SAT {400..1600} --> 400
▶▶ Enter SAT {400..1600} --> 1600
▶▶ Enter SAT {400..1600} --> 1599
▶▶ Enter your gender {M/F} --> F
▶▶ Enter your last name --> Jones
```

```
Ms. Jones, you are admitted!
```

```
----jGRASP: operation complete.
```

```
1 # LogicError03.py
2 # This program is also similar to program InputProtection03.py,
3 # but now there are 2 logic errors because DeMorgan's Law was
4 # not followed when the <not> was distributed in the <while>
5 # statements. The result is the first loop will never execute,
6 # and the second loop will repeat forever.
7
8
9 print()
10
11 sat = 0
12 while sat < 400 and sat > 1600:
13     sat = eval(input("Enter SAT {400..1600} --> "))
14
15 gender = ''
16 while gender != 'M' or gender != 'F':
17     gender = input("Enter your gender {M/F} --> ")
18
19 lastName = input("Enter your last name --> ")
20 print()
21
22 if gender == 'M':
23     print("Mr.",lastName,end = ", ")
24 if gender == 'F':
25     print("Ms.",lastName,end = ", ")
26
27 if sat >= 1100:
28     print("you are admitted!")
29 else:
30     print("you are not admitted.")
```



```
1 # LogicError03.py
2 # This program is
3 # but now there a
4 # not followed wh
5 # statements. Th
6 # and the second
```

```
7
8
9 print()
```

```
10
11 sat = 0
```

```
12 while sat < 400 and sat > 1600:
```

```
13     sat = eval(input("Enter SAT {400..1600} --> "))
```

```
14
```

```
15 gender = ''
```

```
16 while gender != 'M' or gender != 'F':
```

```
17     gender = input("Enter your gender {M/F} --> ")
```

```
18
```

```
19 lastName = input("Enter your last name --> ")
```

```
20 print()
```

```
21
```

```
22 if gender == 'M':
```

```
23     print("Mr.", lastName, end = ", ")
```

```
24 if gender == 'F':
```

```
25     print("Ms.", lastName, end = ", ")
```

```
26
```

```
27 if sat >= 1100:
```

```
28     print("you are admitted!")
```

```
29 else:
```

```
30     print("you are not admitted.")
```

```
----jGRASP exec: python LogicError03
```

```
Enter your gender {M/F} --> Z
```

```
Enter your gender {M/F} --> #
```

```
Enter your gender {M/F} --> F
```

```
Enter your gender {M/F} --> M
```

```
Enter your gender {M/F} --> M
```

```
: : : : : :
```