

# Operators, Variables, and Expressions

Department of Computer Sciences  
University of Wisconsin-Madison

## Readings:

Chapters 1 and 2 of Think Python,  
Chapters 2 and 3 of Python for Everybody

**Additional readings:**  
Computer terminology

# What if I have to miss class?

It's summer, and family obligations/vacation/getting sick happens. Here's what you should do:

- Review lecture materials — download the template files and worksheets and work through them on your own
- If you have to miss lab, complete it on your own. Try to find a project partner who will work with you over Zoom.
- Proactively attend office hours
- Work on class material consistently — a few hours a day rather than cramming over a single day

# **Part 1: Operators**

## **Part 2: Expressions and Variables**



# Learning Objectives

- Run Python code using:
  - Command line
  - Jupyter Notebook

## Evaluate:

- numeric expressions containing mathematical operators (e.g., “+” and “-“)
- string expressions containing string operators and escape characters

## Differentiate:

- behavior of the /, //, and % operators

## Recognize examples of different Python data types:

- int, float, str, bool

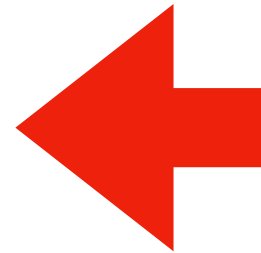
## Evaluate:

- expressions containing comparison operators (e.g., “==” and “>”)
- Boolean expressions containing the operators “and”, “or”, “not”
- mixed expressions using the correct order of operations (precedence)

# Today's Outline

## Software

- Interpreters
- Notebooks



## *Demos*

## Operator Precedence

## *Demos*

## Boolean Logic

## *Demos*

# What you need to write/run code

## An interpreter

- Python 3 (not 2!)
- some extra packages (comes with anaconda installation)
- runs Python code

## Jupyter Notebooks

- comes with anaconda installation
- acts like both interpreter and editor (type and save Python code)

# Interpreter

A program that runs a program

- Translates something the human likes (nice Python code) to something the machine likes (ONEs and ZEROs)

# Jupyter Notebooks

notebooks breakup code into  
"cells" containing Python code

...

```
In [35]: #q22
df = pd.read_sql("""
SELECT continent, count() as num_countries
from countries_table
group by continent
ORDER BY num_countries, continent
""", conn).set_index("continent")

ax = df.sort_index().plot.bar()
ax.set_ylabel("number of countries")
ax.set_xlabel("")
```

A Notebook is a file that contains code and other things  
(e.g., documentation, images, tables, etc.)



# Jupyter Notebooks

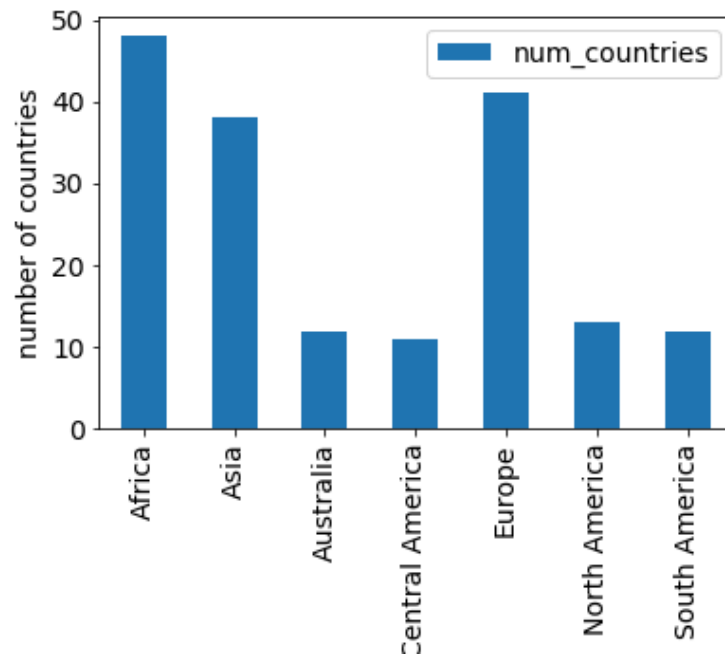
notebooks breakup code into  
"cells" containing Python code

...

```
In [35]: #q22
df = pd.read_sql("""
SELECT continent, count() as num_countries
from countries_table
group by continent
ORDER BY num_countries, continent
""", conn).set_index("continent")

ax = df.sort_index().plot.bar()
ax.set_ylabel("number of countries")
ax.set_xlabel("")
```

Out[35]: Text(0.5, 0, '')



visuals produced by the  
code are embedded in the Notebook

**.ipynb** (Interactive Python Notebook) files are not easy to open in a regular text editor

# 3 ways we'll run Python

## I. interactive mode

Quick syntax check

```
ty-mac:~$ python
Python 3.9.7 (default, Sep 16 2021, 16:59:28)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> 1 + 1
2
```

*triple arrows mean Python code runs as you type it*

# 3 ways we'll run Python

## 1. interactive mode

```
ty-mac:~$ python
Python 3.9.7 (default, Sep 16 2021, 16:59:28)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> 1 + 1
2
```

*triple arrows mean Python code runs as you type it*

## 2. script mode

Run auto-grader tests

*the interpreter program is named "python"; run it*

```
ty-mac:~$ python test.py
```

*the name of the file containing your code (called a "script")  
is passed as an argument to the python program*

# 3 ways we'll run Python

## 1. interactive mode

```
ty-mac:~$ python
Python 3.9.7 (default, Sep 16 2021, 16:59:28)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> 1 + 1
2
```

*triple arrows mean Python code runs as you type it*

## 2. script mode

```
ty-mac:~$ python test.py
```

*the interpreter program is named "python"; run it*

*the name of the file containing your code (called a "script")  
is passed as an argument to the python program*

## 3. notebook "mode"

```
ty-mac:~$ jupyter notebook
```

*open Jupyter in a web browser*

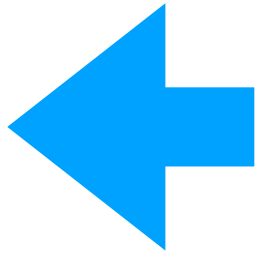
**we'll do most work in notebooks this semester**

# Today's Outline

## Software

- Interpreters
- Notebooks

*Demos*



## Operator Precedence

*Demos*

## Boolean Logic

*Demos*

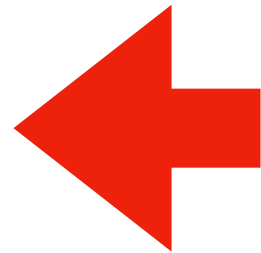
# Today's Outline

## Software

- Interpreters
- Notebooks

## *Demos*

## Operator Precedence



## *Demos*

## Boolean Logic

## *Demos*

# Order of Simplification

Python works by simplifying, applying one operator at a time

$3 * 3 + 2 * 2 + 16 ** (1/2)$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent `**` operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$$3 * 3 + 2 * 2 + 16 ** (1/2)$$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent \*\* operator)



# Order of Simplification

Python works by simplifying, applying one operator at a time

$3 * 3 + 2 * 2 + 16 ** (1/2)$

$3 * 3 + 2 * 2 + 16 ** (0.5)$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent `**` operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$3 * 3 + 2 * 2 + 16 ** (1/2)$

$3 * 3 + 2 * 2 + 16 ** (0.5)$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent `**` operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$3 * 3 + 2 * 2 + 16 ** (1/2)$

$3 * 3 + 2 * 2 + 16 ** (0.5)$

$3 * 3 + 2 * 2 + 4$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent `**` operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$3 * 3 + 2 * 2 + 16 ** (1/2)$

$3 * 3 + 2 * 2 + 16 ** (0.5)$

$3 * 3 + 2 * 2 + 4$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent **\*\*** operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$3 * 3 + 2 * 2 + 16 ** (1/2)$

$3 * 3 + 2 * 2 + 16 ** (0.5)$

$3 * 3 + 2 * 2 + 4$

$9 + 2 * 2 + 4$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent `**` operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$3 * 3 + 2 * 2 + 16 ** (1/2)$

$3 * 3 + 2 * 2 + 16 ** (0.5)$

$3 * 3 + 2 * 2 + 4$

$9 + 2 * 2 + 4$

## Rules

- First work within parentheses
- **Do higher precedence first**
- Break ties left to right (exception: exponent **\*\*** operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$$3 * 3 + 2 * 2 + 16 ** (1/2)$$

$$3 * 3 + 2 * 2 + 16 ** (0.5)$$

$$3 * 3 + 2 * 2 + 4$$

$$9 + 2 * 2 + 4$$

$$9 + 4 + 4$$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent \*\* operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$3 * 3 + 2 * 2 + 16 ** (1/2)$

$3 * 3 + 2 * 2 + 16 ** (0.5)$

$3 * 3 + 2 * 2 + 4$

$9 + 2 * 2 + 4$

$9 + 4 + 4$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent **\*\*** operator)



# Order of Simplification

Python works by simplifying, applying one operator at a time

$$3 * 3 + 2 * 2 + 16 ** (1/2)$$

$$3 * 3 + 2 * 2 + 16 ** (0.5)$$

$$3 * 3 + 2 * 2 + 4$$

$$9 + 2 * 2 + 4$$

$$9 + 4 + 4$$

$$13 + 4$$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent \*\* operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$$3 * 3 + 2 * 2 + 16 ** (1/2)$$

$$3 * 3 + 2 * 2 + 16 ** (0.5)$$

$$3 * 3 + 2 * 2 + 4$$

$$9 + 2 * 2 + 4$$

$$9 + 4 + 4$$

$$13 + 4$$

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent `**` operator)

# Order of Simplification

Python works by simplifying, applying one operator at a time

$$3 * 3 + 2 * 2 + 16 ** (1/2)$$

$$3 * 3 + 2 * 2 + 16 ** (0.5)$$

$$3 * 3 + 2 * 2 + 4$$

$$9 + 2 * 2 + 4$$

$$9 + 4 + 4$$

$$13 + 4$$

**17**

## Rules

- First work within parentheses
- Do higher precedence first
- Break ties left to right (exception: exponent \*\* operator)

# Operator Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x
multiply/divide	*, /, //, %
add/subtract	+, -
comparison	==, !=, <, <=, >, >=
boolean stuff	not
...	and
...	or

simplify first

simplify last\*

these are the ones you should be learning at this point  
in the semester  
(there are a few more not covered now)

\* one exception is an optimization  
known as "short circuiting"

# Operator Precedence

		What is it?	Python Operator	
Mathematical		exponents	**	simplify first
		signs	+x, -x	
		multiply/divide	*, /, //, %	
		add/subtract	+, -	
		comparison	==, !=, <, <=, >, >=	
Logic		boolean stuff	not	simplify last*
		...	and	
		...	or	

these are the ones you should be learning at this point  
in the semester  
(there are a few more not covered now)

\* one exception is an optimization  
known as "short circuiting"

# Today's Outline

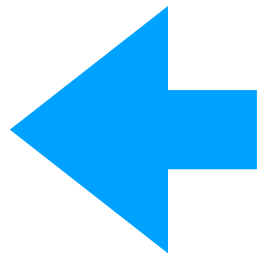
## Software

- Interpreters
- Notebooks

## *Demos*

## Operator Precedence

## *Demos*



## Boolean Logic

## *Demos*

# Today's Outline

## Software

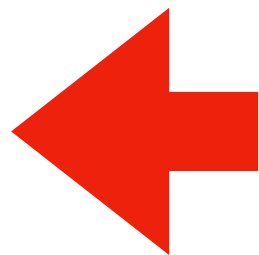
- Interpreters
- Notebooks

## *Demos*

## Operator Precedence

## *Demos*

## Boolean Logic

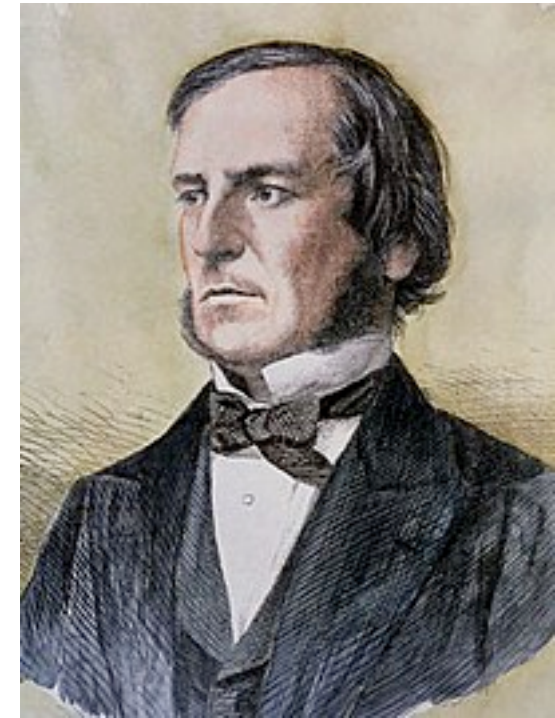


## *Demos*

# Boolean Logic

The logic of truth:

- Named after George Boole
- Two values: True and False
- Three operators: **and**, **or**, and **not**

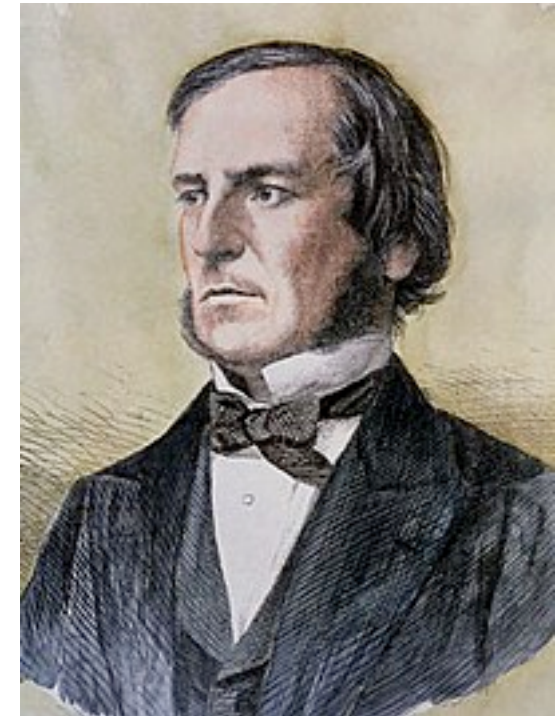




# Boolean Logic

The logic of truth:

- Named after George Boole
- Two values: True and False
- Three operators: **and**, **or**, and **not**



## AND

	False	True
False	False	False
True	False	True

## OR

	False	True
False	False	True
True	True	True

## NOT

False	True
True	False

It's a Saturday **AND**  
we're attending CS 220 lecture

**AND**

	False	True
False	False	False
True	False	True

**OR**

	False	True
False	False	True
True	True	True

**NOT**

False	True
True	False

FALSE!

It's a Saturday **AND**

we're attending CS 220 lecture

**AND**

	False	True
False	False	False
True	False	True

**OR**

	False	True
False	False	True
True	True	True

**NOT**

	False	True
False	True	False

Project I is due on Friday  
**OR** I'll eat my hat



**AND**

	False	True
False	False	False
True	False	True

**OR**

	False	True
False	False	True
True	True	True

**NOT**

	False	True
False	True	False
True	False	True

TRUE!

Project I is due on Friday  
OR I'll eat my hat



AND

	False	True
False	False	False
True	False	True

OR

	False	True
False	False	True
True	True	True

NOT

False	True
True	False

Control Flow: Remember that conditionals and loops *sometimes* do something.  
We'll use bool logic a LOT to control when we do/don't.

AND

	False	True
False	False	False
True	False	True

OR

	False	True
False	False	True
True	True	True

NOT

False	True
True	False

# Today's Outline

## Software

- Interpreters
- Notebooks

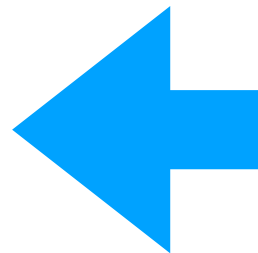
## *Demos*

## Operator Precedence

## *Demos*

## Boolean Logic

## *Demos*



Part 1: Operators

## **Part 2: Expressions and Variables**





# Learning Objectives

Evaluate expressions by identifying:

- operators and operands
- literal values and variables
- correct order of operations

Write correct Boolean expressions

- containing Boolean operators “or” and “and”

Write assignment statements

- with variables following proper naming rules

Define, give examples of, and identify 3 kinds of errors

- Syntax, runtime, and semantic

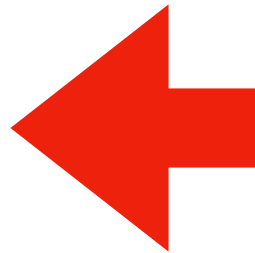
Write code to perform computations with

- int, float, string, and bool types

# Today's Outline

## Review

- Operator Precedence



Expressions, Variables, and Assignments

## Demos

Bugs



## Demos

Naming variables

## Demos

Unordered

What is it?	Python Operator
comparison	==, !=, <, <=, >, >=
signs	+x, -x
AND	and
add/subtract	+, -
exponents	**
NOT	not
OR	or
multiply/divide	*, /, //, %

Ordered by Precedence

What is it?	Python Operator

simplify first

simplify last

Unordered

What is it?	Python Operator
comparison	==, !=, <, <=, >, >=
signs	+x, -x
AND	and
add/subtract	+, -
NOT	not
OR	or
multiply/divide	*, /, //, %

Ordered by Precedence

What is it?	Python Operator
exponents	**

simplify first

simplify last

Unordered

What is it?	Python Operator
comparison	==, !=, <, <=, >, >=
AND	and
add/subtract	+, -
NOT	not
OR	or
multiply/divide	*, /, //, %

Ordered by Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x

simplify first

simplify last

Unordered

What is it?	Python Operator
comparison	==, !=, <, <=, >, >=
AND	and
add/subtract	+, -
NOT	not
OR	or

Ordered by Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x
multiply/divide	*, /, //, %

simplify first

simplify last

Unordered

What is it?	Python Operator
comparison	==, !=, <, <=, >, >=
AND	and
NOT	not
OR	or

Ordered by Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x
multiply/divide	*, /, //, %
add/subtract	+, -

simplify first

simplify last

Unordered

What is it?	Python Operator
AND	and
NOT	not
OR	or

Ordered by Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x
multiply/divide	*, /, //, %
add/subtract	+, -
comparison	==, !=, <, <=, >, >=

simplify first

simplify last



Unordered

What is it?	Python Operator
AND	and
OR	or

Ordered by Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x
multiply/divide	*, /, //, %
add/subtract	+, -
comparison	==, !=, <, <=, >, >=
NOT	not

simplify first

simplify last

Unordered

What is it?	Python Operator
OR	or

Ordered by Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x
multiply/divide	*, /, //, %
add/subtract	+, -
comparison	==, !=, <, <=, >, >=
NOT	not
AND	and

simplify first

simplify last

Unordered

What is it?	Python Operator

Ordered by Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x
multiply/divide	*, /, //, %
add/subtract	+, -
comparison	==, !=, <, <=, >, >=
NOT	not
AND	and
OR	or

simplify first

simplify last

Unordered

What is it?	Python Operator

Ordered by Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x
multiply/divide	*, /, //, %
add/subtract	+, -
comparison	==, !=, <, <=, >, >=
NOT	not
AND	and
OR	or

simplify first

10 - -2 // 3

simplify last

Unordered

What is it?	Python Operator

Ordered by Precedence

What is it?	Python Operator
exponents	**
signs	+x, -x
multiply/divide	*, /, //, %
add/subtract	+, -
comparison	==, !=, <, <=, >, >=
NOT	not
AND	and
OR	or

simplify first

simplify last

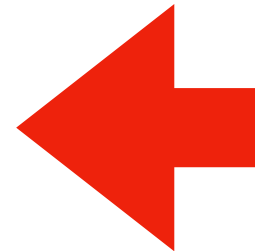
1 + 1 == 2 or 3 \*\* 10000000 > 2 \*\* 20000000

logical operators  
can "short circuit"

# Today's Outline

Review

Expressions, Variables, and Assignments



*Demos*

Bugs



*Demos*

Naming variables

*Demos*

# Expressions

Expressions are a mix of operators and operands. For example:

$5 + 5$

$(8/2) ** 2 * 3.14$

$3 * 3 > 4 + 4$

$3 \% 2 == 0$  or  $3 \% 2 == 1$

# Expressions

Expressions are a mix of operators and operands. For example:

$5 + 5$

$(8/2) ** 2 * 3.14$

$3 * 3 > 4 + 4$

$3 \% 2 == 0$  or  $3 \% 2 == 1$

Each of these operands is an example of a *literal*: a fixed value



# Expressions

Expressions are a mix of operators and operands. For example:

$x + y$

$(\text{diameter}/2) ** 2 * \text{pi}$

$\text{value1} * \text{value1} > \text{value2} + \text{value2}$

$\text{num} \% 2 == 0$  or  $\text{num} \% 2 == 1$

An operand may also be a *variable*: not fixed

# Expressions

Expressions are a mix of operators and operands. For example:

$x + y$

(diameter

value l \*

num % 2

Quick Test! Circle the **literals** (others are **variables**)

1. 0

2. zero

3. num l

4. True

5. hello

6. "goodbye"

An operand may also be a **variable**: not fixed

# Expressions

Expressions are a mix of operators and operands. For example:

$x + y$

(diameter

value l \*

num % 2

Quick Test! Circle the **literals** (others are **variables**)

1. 0

2. zero

3. num l

4. True

5. hello

6. "goodbye"

An operand may also be a **variable**: not fixed

# Expressions

Expressions are a mix of operators and operands. For example:

$x + y$

(diameter

value  $\times$

num  $\% 2$

Quick Test! Circle the **literals** (others are **variables**)

1. 0

2. zero

3. num1

4. True

5. hello

6. "goodbye"

An operand may also be a **variable**: not fixed

# Expressions

Expressions are a mix of operators and operands. For example:

$x + y$

(diameter

value  $l * 2$

num  $\% 2$

Quick Test! Circle the **literals** (others are **variables**)

1. 0

2. zero

3. num1

4. True

5. hello

6. "goodbye"

An operand may also be a **variable**: not fixed

# Expressions

Expressions are a mix of operators and operands. For example:

$x + y$

(diameter

value l \*

num % 2

Quick Test! Circle the **literals** (others are **variables**)

1. 0

2. zero

3. num l

4. True

5. hello

6. "goodbye"

An operand may also be a **variable**: not fixed

How do we put a value in a variable?

# Assignment

An **assignment** computes an expression (maybe a simple one) and puts the result in a variable:

**x** + **y**

(**diameter**/2) \*\* 2 \* **pi**

**value1** \* **value1** > **value2** + **value2**

**num** % 2 == 0 or **num** % 2 == 1

# Assignment

An **assignment** computes an expression (maybe a simple one) and puts the result in a variable:

$x + y$

$(\text{diameter}/2) ** 2 * \text{pi}$

$\text{value1} * \text{value1} > \text{value2} + \text{value2}$

$\text{num} \% 2 == 0$  or  $\text{num} \% 2 == 1$



# Assignment

An **assignment** computes an expression (maybe a simple one) and puts the result in a variable:

= **x** + **y**

= (**diameter**/2) \*\* 2 \* **pi**

= **value1** \* **value1** > **value2** + **value2**

= **num** % 2 == 0 or **num** % 2 == 1

# Assignment

An **assignment** computes an expression (maybe a simple one) and puts the result in a variable:

```
total = x + y
```

```
area = (diameter/2) ** 2 * pi
```

```
is_bigger = value1 * value1 > value2 + value2
```

```
is_even_or_odd = num % 2 == 0 or num % 2 == 1
```

# Assignment

An **assignment** computes an expression (maybe a simple one) and puts the result in a variable:

**total** = **x** + **y**

**area** = (**diameter**/2) \*\* 2 \* **pi**

**is\_bigger** = **value1** \* **value1** > **value2** + **value2**

**is\_even\_or\_odd** = **num** % 2 == 0 or **num** % 2 == 1

Expression

# Assignment

An **assignment** computes an expression (maybe a simple one) and puts the result in a variable:

**total** = **x** + **y**

**area** = (**diameter**/2) \*\* 2 \* **pi**

**is\_bigger** = **value1** \* **value1** > **value2** + **value2**

**is\_even\_or\_odd** = **num** % 2 == 0 or **num** % 2 == 1



Assignment Operator

Expression

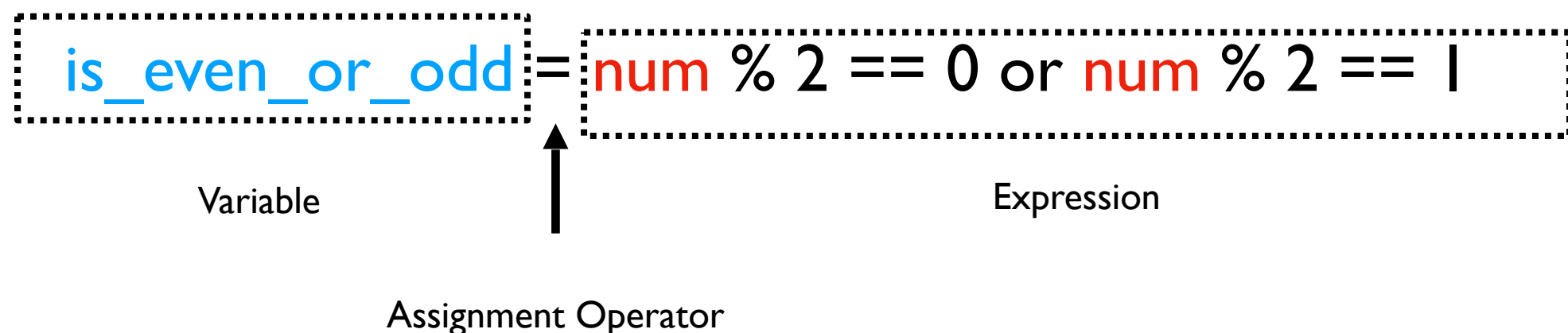
# Assignment

An **assignment** computes an expression (maybe a simple one) and puts the result in a variable:

**total** = **x** + **y**

**area** = (**diameter**/2) \*\* 2 \* **pi**

**is\_bigger** = **value1** \* **value1** > **value2** + **value2**

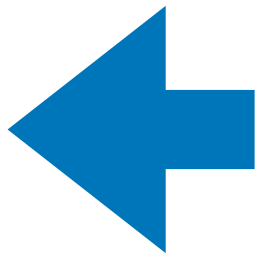


# Today's Outline

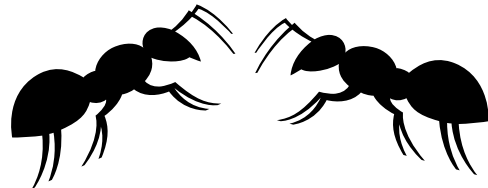
Review

Expressions, Variables, and Assignments

*Demos*



Bugs



*Demos*

Naming variables

*Demos*

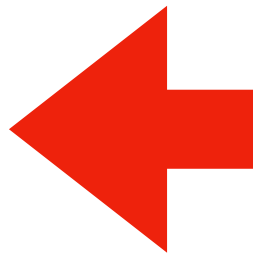
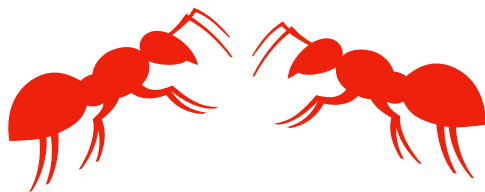
# Today's Outline

Review

Expressions, Variables, and Assignments

*Demos*

Bugs



*Demos*

Naming variables

*Demos*

# Categories of Errors

1

dog cat the of chase any  
[word soup, not grammatically sensible]

2

3



# Categories of Errors

1

## Syntax Error

- It never makes sense in any context; Python doesn't even run

● 5 = x

2

3

# Categories of Errors

1

## Syntax Error

- It never makes sense in any context; Python doesn't even run
- `5 = x`

2

this sentence is false

[grammatical, but my head explodes if I think about it]

3

# Categories of Errors

1

## Syntax Error

- It never makes sense in any context; Python doesn't even run

- `5 = x`

2

## Runtime Error

- Need to run to find out whether it will crash
- Appears with different names (TypeError, ZeroDivisionError, etc)

- `x = 5 / 0`

3

# Categories of Errors

1

## Syntax Error

- It never makes sense in any context; Python doesn't even run

- `5 = x`

2

## Runtime Error

- Need to run to find out whether it will crash
- Appears with different names (TypeError, ZeroDivisionError, etc)

- `x = 5 / 0`

3

one week is 10 days long  
[grammatical, coherent, but incorrect]

# Categories of Errors

1

## Syntax Error

- It never makes sense in any context; Python doesn't even run

- `5 = x`

2

## Runtime Error

- Need to run to find out whether it will crash
- Appears with different names (TypeError, ZeroDivisionError, etc)

- `x = 5 / 0`

3

## Semantic Error

- It runs with no error, but you get the wrong answer

- `square_area = square_side * 2`

# Categories of Errors

1

## Syntax Error

- It never makes sense in any context; Python doesn't even run

• `5 = x`

2

## Runtime Error

- Need
  - Appea
- what kind of error is the worst?** etc)

• `x = 5 / 0`

3

## Semantic Error

- It runs with no error, but you get the wrong answer

• `square_area = square_side * 2`

# Today's Outline

Review

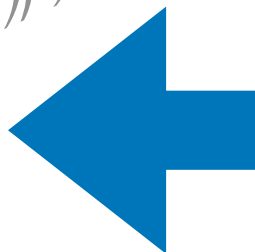
Expressions, Variables, and Assignments

*Demos*

Bugs



*Demos*



Naming variables

*Demos*

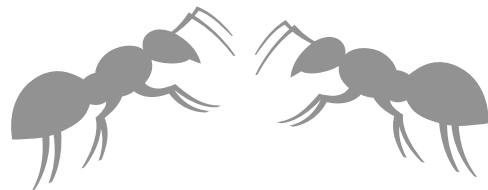
# Today's Outline

Review

Expressions, Variables, and Assignments

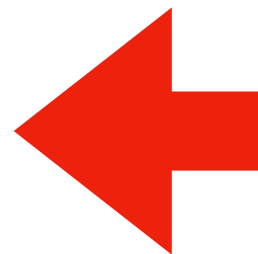
*Demos*

Bugs



*Demos*

Naming variables



*Demos*



# What Variable Names are Allowed?

`1st_score = 100` [bad variable]

`score_1 = 100` [good variable]

`firstScore = 100` [not a recommended variable]  
`= 100` [recommended variable]

`first_score`

current rules are quite complex:

<https://www.python.org/dev/peps/pep-3131>

please don't use camel case:

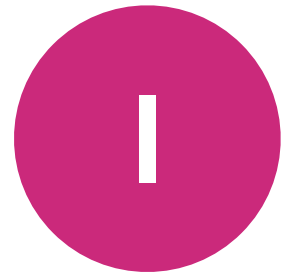
<https://www.python.org/dev/peps/pep-0008/>

Python 3 has become friendlier to non-English programmers

`quero_café = True`

← this is allowed, and  
different than "e"

# Rules for naming variables



Only use letters a-z (upper and lower), numbers, and underscores



Don't start with a number



Don't use Python keywords (e.g., and, False, etc)

For 220, you may use only variables containing English alphabets

# Rules for naming variables

1

Only use letters a-z (upper and lower), numbers, and underscores

2

Don't start with a number

3

Don't use Python keywords (e.g., and, False, etc)

GOOD:

cs220  
CS220  
cs\_220  
\_cs220

BAD:

220class  
and  
pi3.14  
x!

*what rules are violated?*

# Rules for naming variables

1

Only use letters a-z (upper and lower), numbers, and underscores

2

Don't start with a number

3

Don't use Python keywords (e.g., and, False, etc)

GOOD:

cs220  
CS220  
cs\_220  
\_cs220

BAD:

220class  
and  
pi3.14  
x!

2

# Rules for naming variables

1

Only use letters a-z (upper and lower), numbers, and underscores

2

Don't start with a number

3

Don't use Python keywords (e.g., and, False, etc)

GOOD:

cs220  
CS220  
cs\_220  
\_cs220

BAD:

220class  
and  
pi3.14  
x!

2

3

# Rules for naming variables

1

Only use letters a-z (upper and lower), numbers, and underscores

2

Don't start with a number

3

Don't use Python keywords (e.g., and, False, etc)

GOOD:

cs220  
CS220  
cs\_220  
\_cs220

BAD:

220class  
and  
pi3.14  
x!

2

3

1

1

# Rules for naming variables

1

Only use letters a-z (upper and lower), numbers, and underscores

2

Don't start with a number

3

Don't use Python keywords (e.g., and, False, etc)

GOOD:

cs220  
CS220  
cs\_220  
\_cs220

BAD:

220class  
and  
pi3.14  
x!

2

3

1

1

# Identifying keywords

3

Don't use Python keywords (e.g., and, False, etc)

*How to figure out if something is a Python keyword?*

- Python keywords turn green in color in jupyter notebook
- If used as a variable, that keyword will no longer work as intended!

GOOD:

```
In [ ]: 1 player
        2 start
        3 hurricane_speed
        4 final_score
```

BAD:

```
In [ ]: 1 print
        2 list
        3 type
        4 bool
```

*Pay attention to green colorization within jupyter notebook*



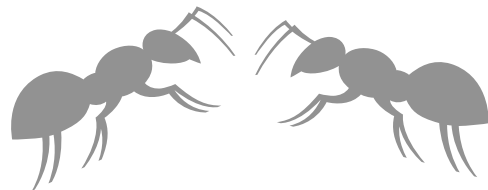
# Today's Outline

Review

Expressions, Variables, and Assignments

*Demos*

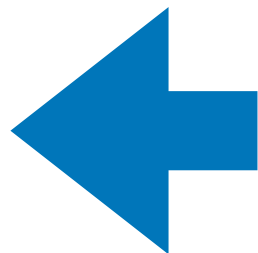
Bugs



*Demos*

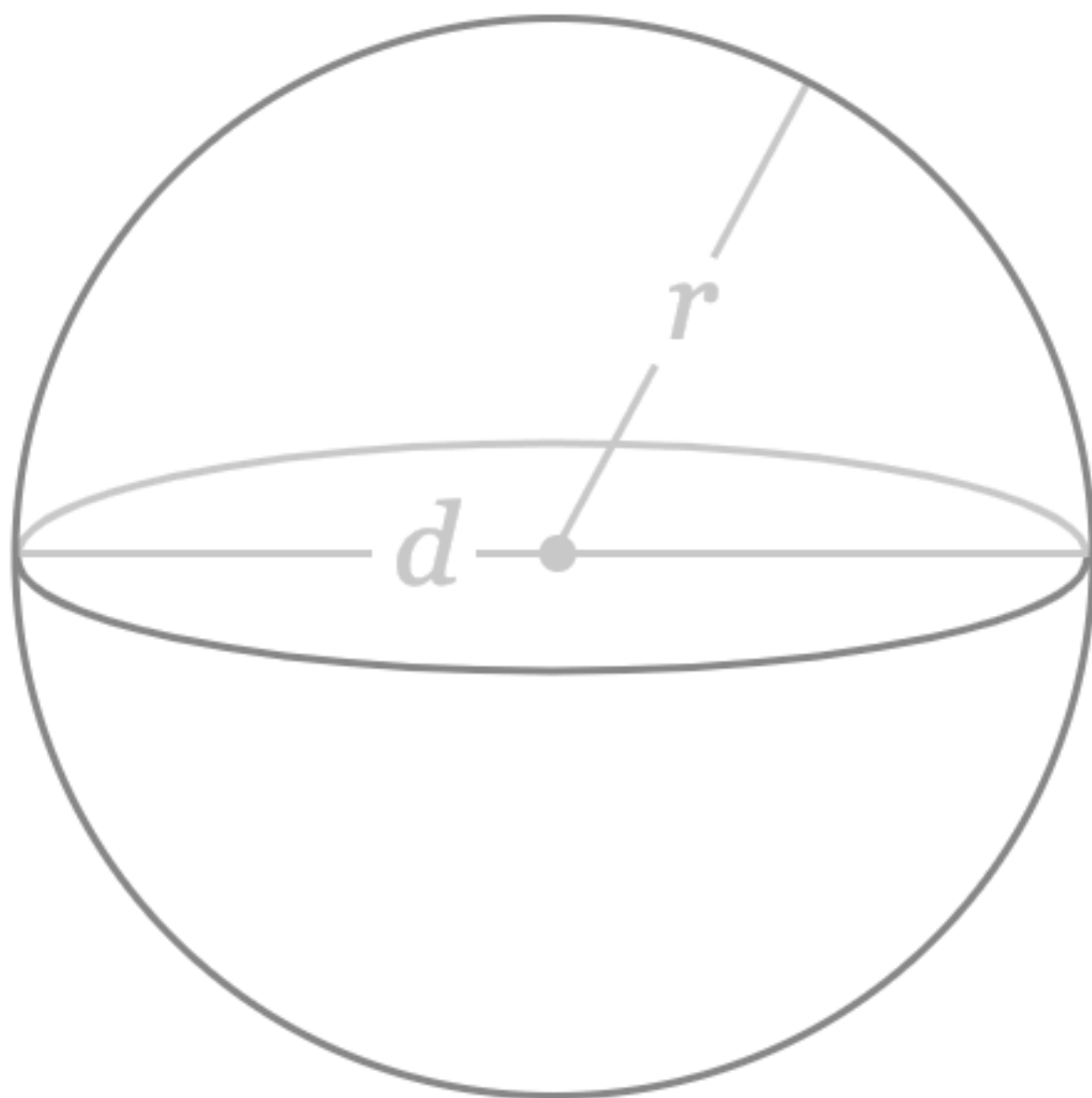
Naming variables

*Demos*



# Practice Problems

# Practice: Sphere Volume

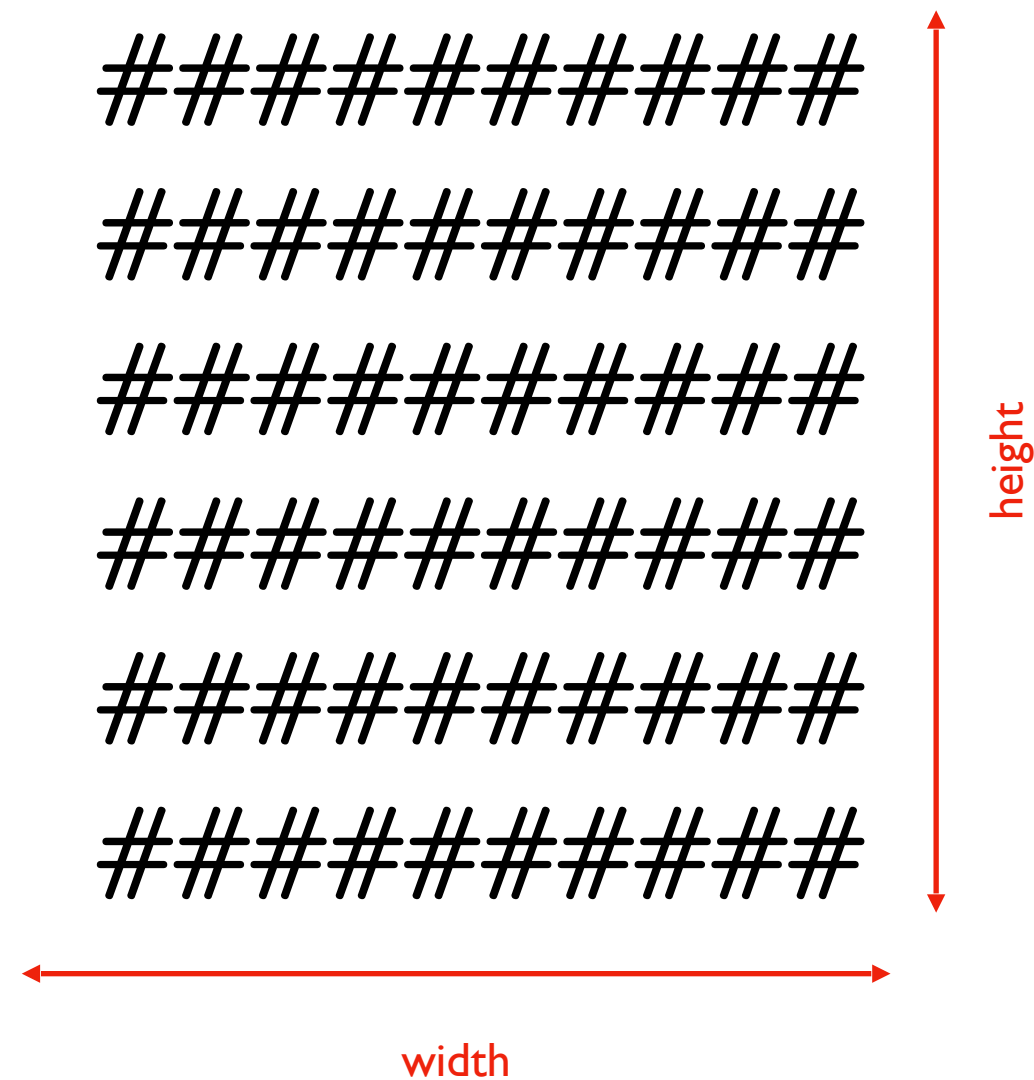


$$V = \frac{4}{3} \pi r^3$$

extension: find radius given a volume

# Practice: Character Art - Block

write some code to draw the following:



# Practice: Quadratic Formula

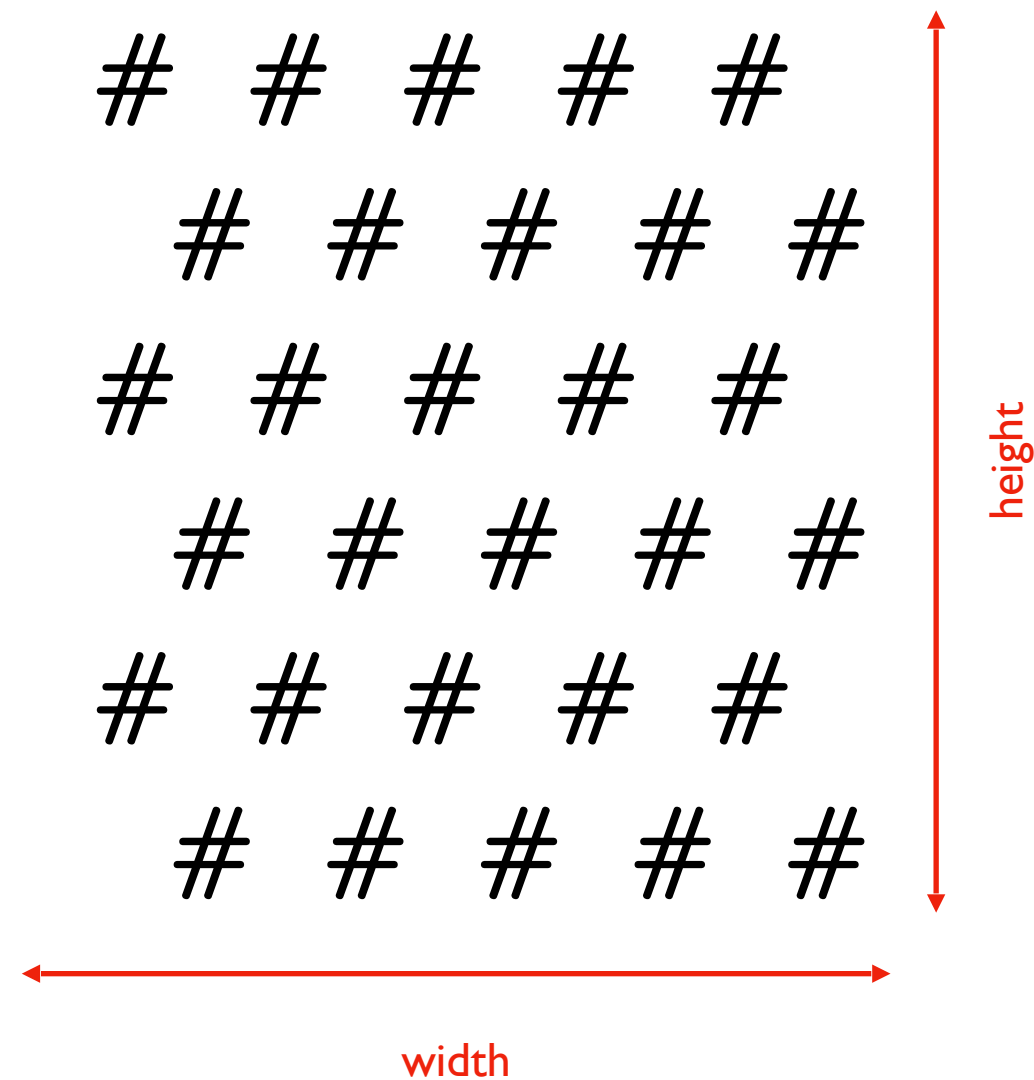
$$ax^2 + bx + c = 0$$

*what values of  $x$  satisfy the above?*

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

# Challenge\*: Checkers

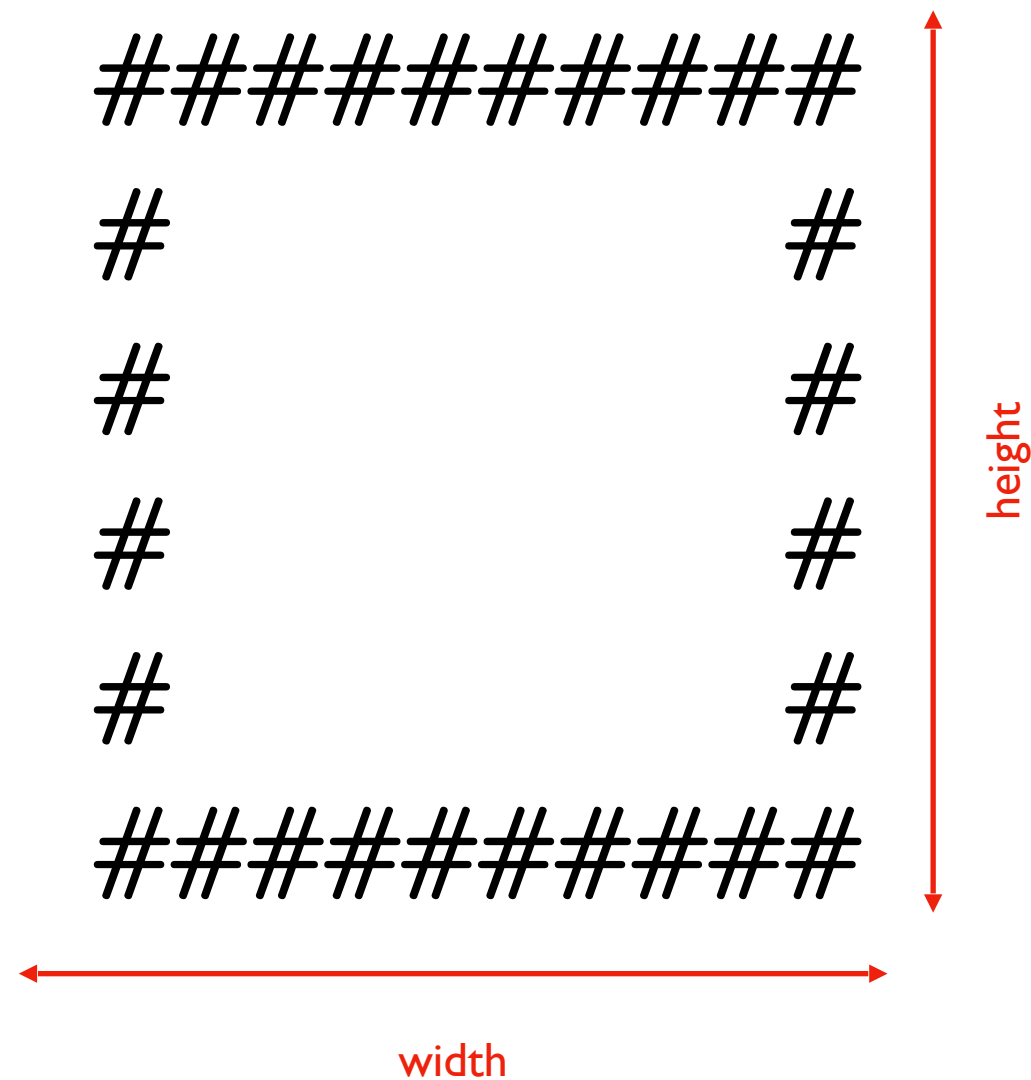
write some code to draw the following:



\* Challenge = beyond what you would be asked to do on an exam

# Challenge: Border

write some code to draw the following:



# Challenge: Snake

write some code to draw the following:

