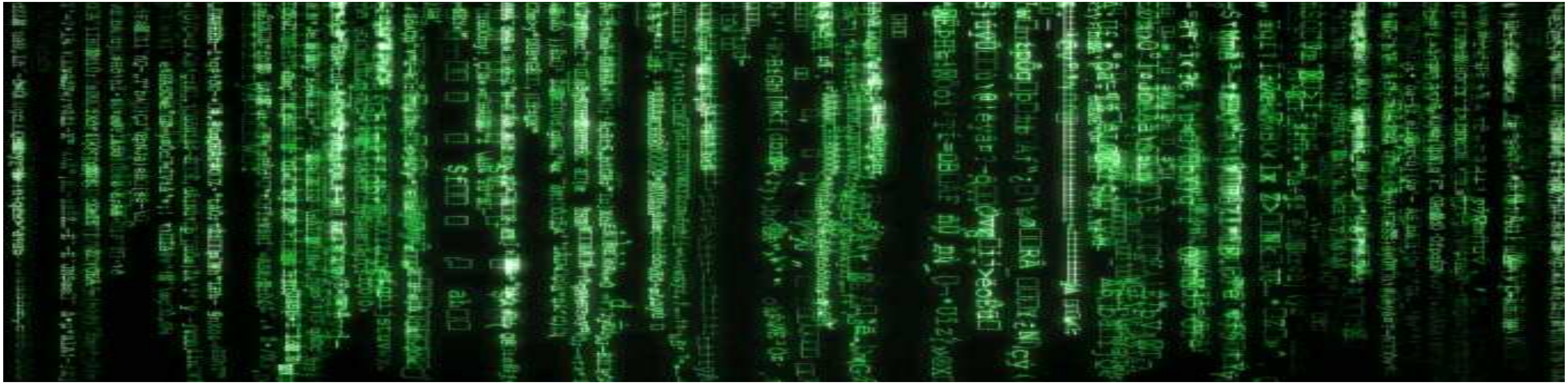


INF 110 Discovering Informatics

Python Expressions (part 1)

NAU
NORTHERN
ARIZONA
UNIVERSITY

Why Python?

- Python is *simple*
- Python is *easy to learn*
- Python is *free*
- Python is a *community*
- Python is a *high-level language*
  - **a lot of tasks are performed** *“behind the scenes”*
  - **eliminates a lot of** *“conceptual overhead”*
  - **still maintains a lot of functionality**
- Python is commonly used in *informatics & data science*
  - *we are not that interested in the exactly* **how** *the computations are performed (at least in this class)*

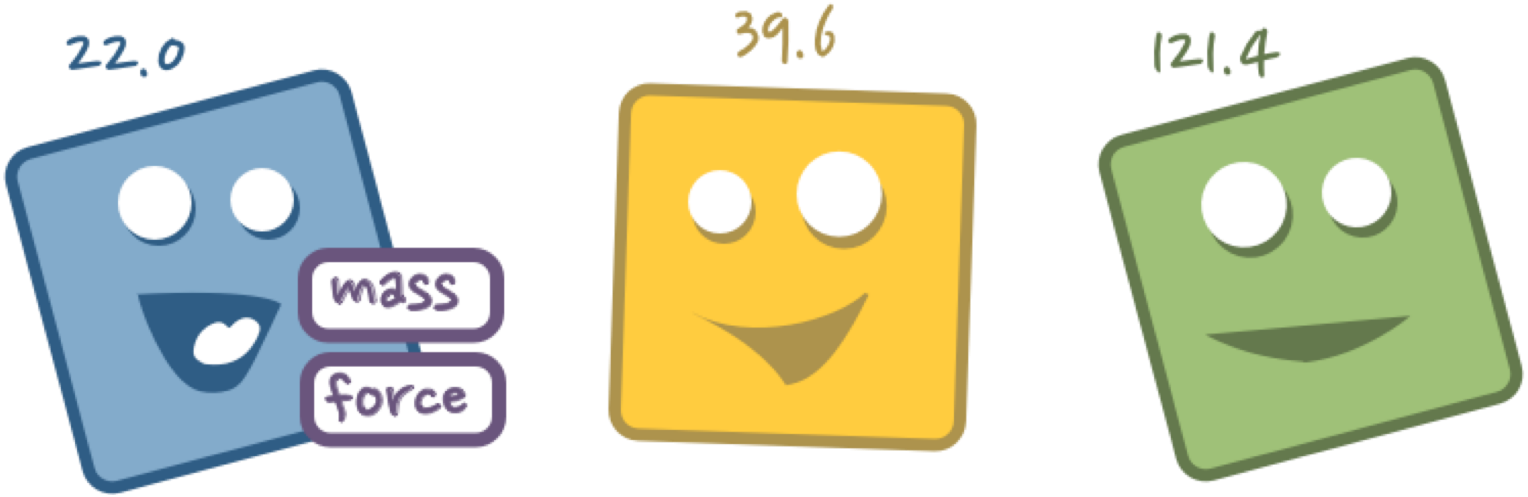
Your first programming principle: Variables

Variable: ***a symbolic representation of a value***

- **Connects a name or a label to one value**
- **Variables can change over time**
- **But,** *a single variable can only have one value at at time.*

22.0
39.6
121.4
mass
force

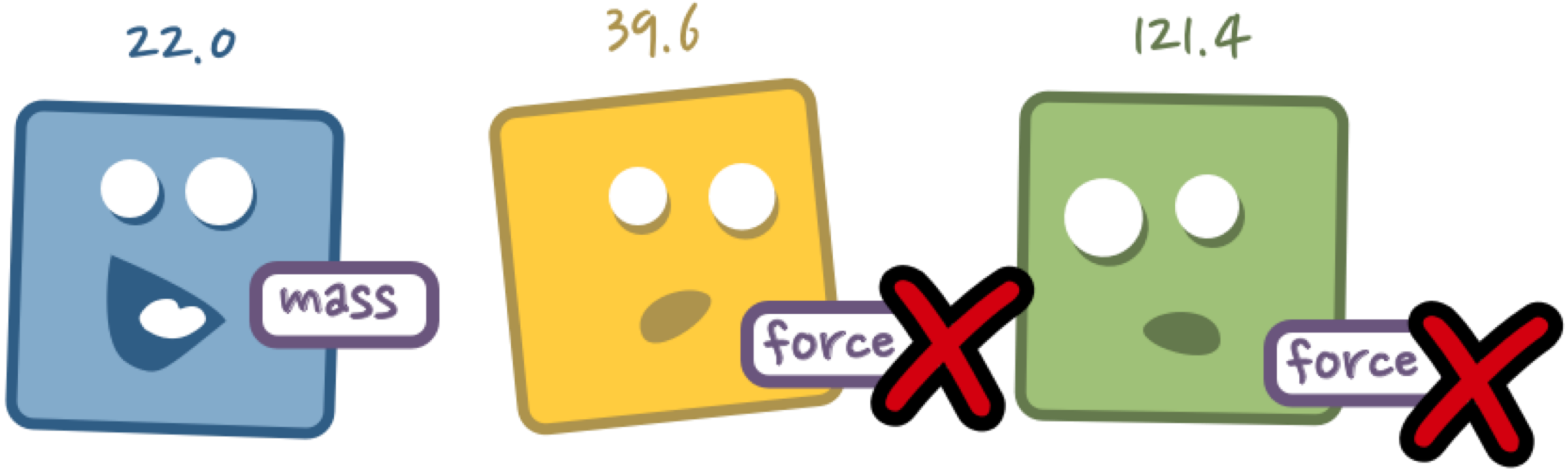
Variables



However, one value could wear multiple labels - meaning that a single value is connected to multiple variables (i.e. both ***mass*** and ***force*** can equal ***22.0***).

- a single person can only have one birthday, but that birthday can be shared by multiple people

Variables

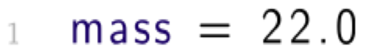


But one important rule at this party is that labels must be **unique** - two values can’t have the same label (i.e., variable name) at once.

- Or, two values can’t occupy the same space at the same time.

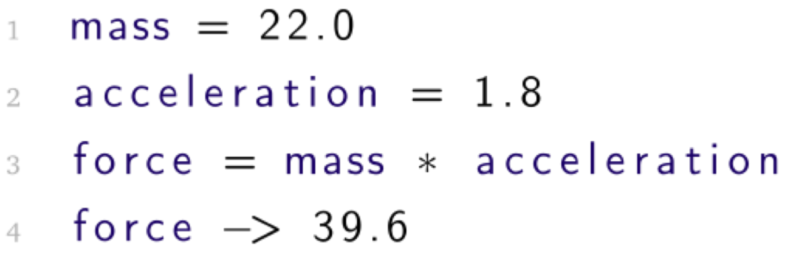
Variable Assignment

Variables are connected to values through through **assignment** - this is how we make a value wear a label.



```
mass = 22.0
```

Consider this example where a new value associated with the variable force (39.6) is derived from a standard physics relationship:



```
mass = 22.0
acceleration = 1.8
force = mass * acceleration
force -> 39.6
```

It’s important to note: the left hand value is always the target for assignment in Python!!!

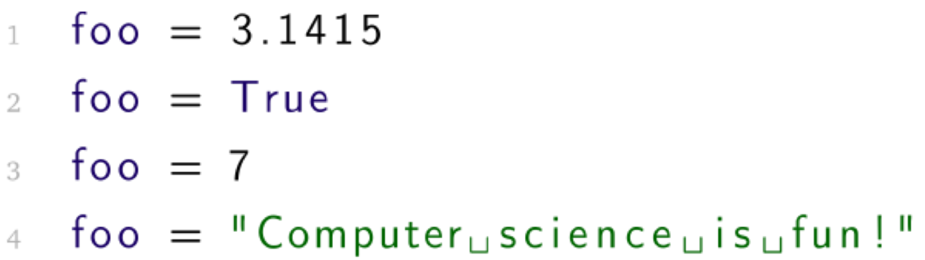
- Entering “4 = x” will break (Python will complain we can’t assign a value to a constant.)

Variable Assignment

Dynamic typing:

- In computer science, there are “data types” (integers, decimals, characters, strings, bytes, files)
- These types are represented differently in the computer’s memory.
- In some languages, you have to assign a data type to a variable – ***but not in Python!!!***
- ***The variable’s value AND its type are dynamic – they change in accordance to the value you assign to the variable.***

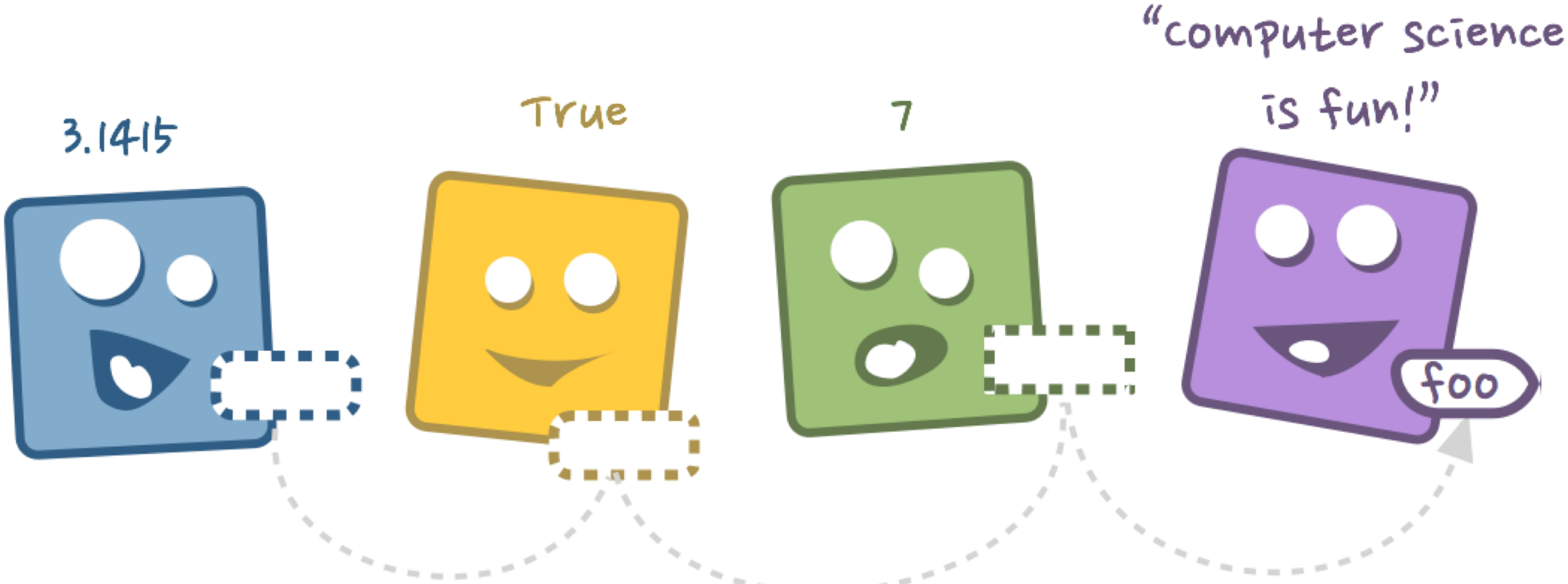
Here we illustrate dynamic typing by letting the variable foo take on four different values and four different corresponding types:



```
foo = 3.1415
foo = True
foo = 7
foo = "Computer␣science␣is␣fun!"
```

(remember, this is legal because you can assign different values to the same variable)

Variable Assignment



Rules for Naming Variables

- variable names are lower case and words are separated with underscores (AKA snake case; e.g., standard_deviation),
- class names (to be covered later) are title case (e.g., ColorMatrix),
- identifiers that begin with one or more underscores have special meaning
- identifiers shouldn’t have the same name as built-in identifiers (e.g., int , float, list, tuple, dir).
  - don’t confuse your variable names with these important Python keywords (“namespace collisions”)

Rules for Naming Variables

- When creating variables, it’s important to follow ***conventions.***
  - ***External conventions*** – i.e. PEP8 coding standard
  - ***Internal conventions*** – maintained by institutions
    - Google and Microsoft have internal conventions.

- You can Google “PEP8” to find out about the external conventions, but here are some good tips:

Choosing **Good** Variable Names

- Is the name consistent with existing naming conventions?
- Does this value have important units (grams, meters, moles, etc.) that are not obvious from its type or usage?
- Does the name unnecessarily use negative logic or other counter intuitive conventions? You should consider using is_enabled instead of is_not_enabled.

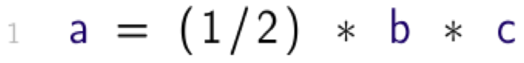
Choosing **Good** Variable Names

- Is the name descriptive?
- If you had seen this variable for the first time would the name make sense?
- Is the name too wordy, long, or redundant?
- Is the name too short or does it use uncommon abbreviations?

Code Should Read Like Poetry

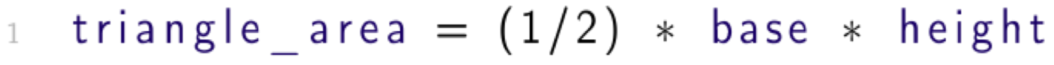
Or at least attempt to be “***self documenting***”: so clear it doesn’t need any further comments or explanation about what its doing.

Consider this perfectly correct piece of code:



```
a = (1/2) * b * c
```

Choose names that reveal the codes purpose:



```
triangle_area = (1/2) * base * height
```

Some Useful Types

- **Internally, your computer represents things as 0s and 1s, but it needs hints as to how those will be interpreted.**
  - **These hints are captured as "*data types*"**

- **Strings** - sequences of alphanumeric characters

  `"Discovering Informatics!”`

- **Integers** – whole numbers

  `1138`

- **Floats** – real numbers

  `3.1415`

Basic *Math* Operations

`x + y`	Addition
`x – y`	Subtraction
`x * y`	Multiplication
`x / y`	Division
`x ** y`	Exponentiation
`abs(x)`	Absolute Value

Note: “times times” for exponentiation, and “absolute values” requires a function*.
Function notation uses an identifier (like with variables) and a set of open-closed parentheses.

- Within the parentheses, we specify a set of arguments to the function.
- Here we’re saying “computer the absolute value of x”

*More on functions later in the course.

Basic *Comparison* Operations

`x < y`	Less than
`x <= y`	Less than or equal to
`x > y`	Greater than
`x >= y`	Greater than or equal to
`x == y`	Equal to

- These always result in a value of True or False
  - AKA “Boolean”
  - A Boolean is another data type which refers to a T or F value.

Warning: Assignment (=) and comparison (==) are different!

Basic Conversions

- Sometimes a value is represented as a certain type, but we want it to represent a different type. We have to perform a “cast” or “typecast”.
- We do this by calling a function that will cast the value to a different type.

• `str(x)`	Convert to a string
• `int(x)`	Convert to an integer
• `float(x)`	Convert to a float

- If you have weird datatypes, this might not always work and you may have to create a different function.

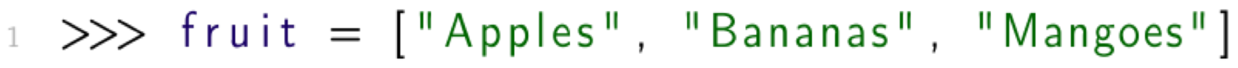
Data Structures

- In computer science, we often use data structures to aggregate or organize our data.
- This allows us to not only store the data, but also access it efficiently without having to name every data point something different.

- A variable can only point to a single value, but that value can be a data structure like a ***list***.

Lists and Dictionaries

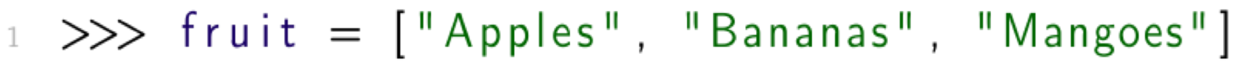
- Python has ***lists*** that are a single point of reference that stores all the bits of information together.
- Python uses square brackets to signify that items belong to a list.
- Items within the brackets are separated by commas.
- We can have as many items as we want in the list.



```
>>> fruit = ["Apples", "Bananas", "Mangoes"]
```

Lists and Dictionaries

- Here’s a list called “fruit” – we assign the variable to a list.
  - Note the quotes – they signify that the items are strings.
  - If there were no quotes, Python would look for variables and break.



```
>>> fruit = ["Apples", "Bananas", "Mangoes"]
```

Lists and Dictionaries

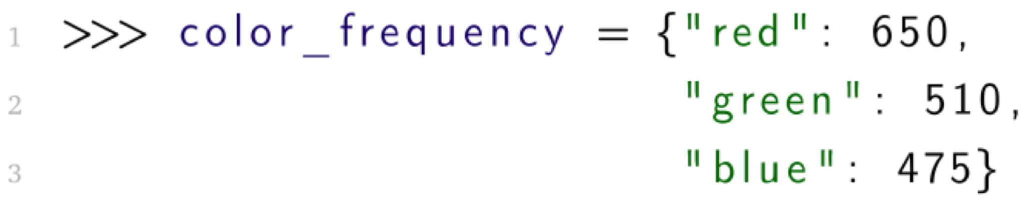
- To access the items in the list, we do so “by index”.
- We say “go to the list called fruit, and grab the element at the first index, index 0”
    - This means that Python will reliably return “apples” every time you ask for the first element.

```
>>> fruit = ["Apples", "Bananas", "Mangoes"]
```

```
>>> fruit[0]
'Apples'
```

Lists and Dictionaries

- The next data structure is a ***dictionary***.
- Instead of being indexed by a number, dictionaries are indexed by a "***key***".
  - The most important concept of a dictionary is the ***key-value pair***.
    - The key defines the name of the object, followed by it's value.
- Python uses curly brackets to signify an object as a dictionary.
  - Within the brackets, key-value pairs are indicated with colons
  - They are separated by commas (except for the last entry).

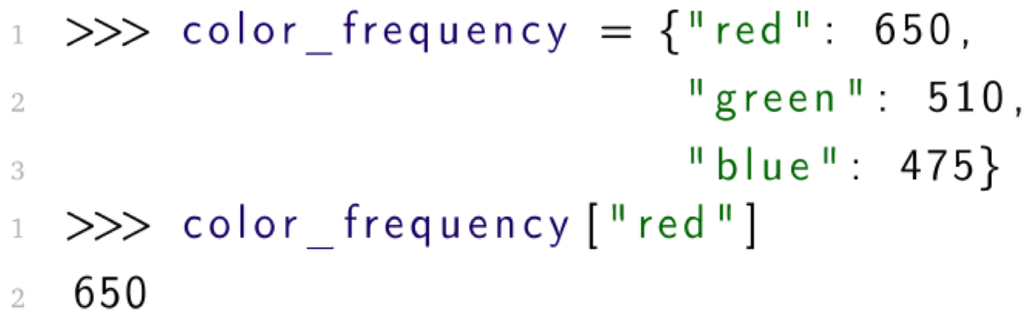


```
>>> color_frequency = {"red": 650,
                       "green": 510,
                       "blue": 475}
```

Lists and Dictionaries

- You can then search through the dictionary by specifying it’s key.
- It’s important to note that the syntax for searching through the dictionary is still square brackets rather than curly ones.

- Think of Python dictionaries like actual dictionaries:
  - Open up a dictionary and find word:definitions, these are like key:value pairs.
  - It’s better to use a dictionary because you can ask “what is the definition of ‘fungible’” rather than ”what is the 156,000th word in this dictionary”

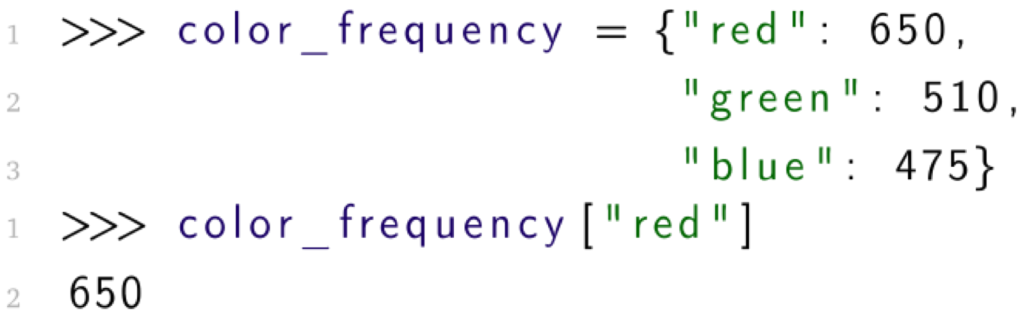


```
>>> color_frequency = {"red": 650,
                       "green": 510,
                       "blue": 475}
>>> color_frequency["red"]
650
```

Lists and Dictionaries

- You can then search through the dictionary by specifying it’s key.
- It’s important to note that the syntax for searching through the dictionary is still square brackets rather than curly ones.

- Think of Python dictionaries like actual dictionaries:
    - Open up a dictionary and find word:definitions, these are like key:value pairs.
    - It’s better to use a dictionary because you can ask “what is the definition of ‘fungible’” rather than ”what is the 156,000th word in this dictionary”



```
>>> color_frequency = {"red": 650,
                       "green": 510,
                       "blue": 475}
>>> color_frequency["red"]
650
```

One key will always map to the same value, but not necessarily in order – so ***don’t access dictionaries by order.***

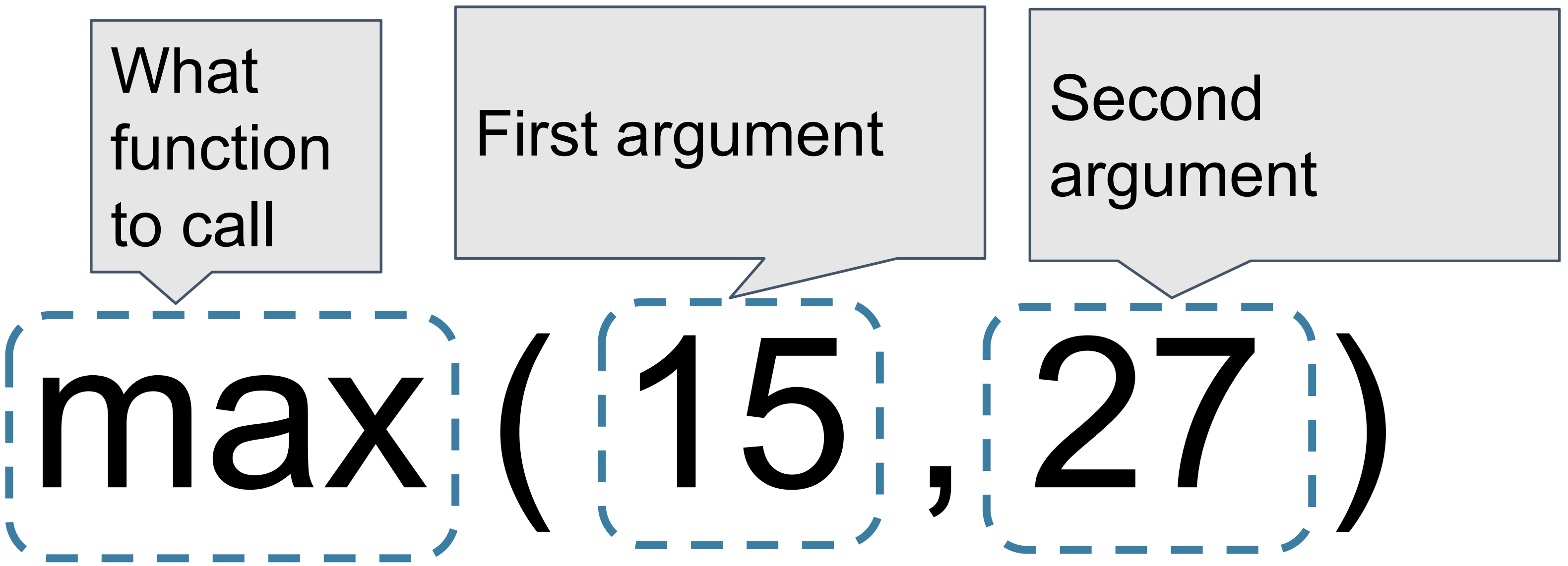
Calling Functions

- Functions are useful when we want to be able to repeatedly use the same block of logic to manipulate certain variables.

- Instead of rewriting the same code over and over again every time we want to use it, we just write it once to a function, and then call it whenever we want.

Calling Functions

- In Python, functions use parentheses notation.
  - Specify the name of the function, then open parentheses, some arguments, the close parentheses. All arguments are separated by commas.
- ***Arguments:*** functions do not understand the outside world and can only understand values that are passed to them as arguments.



What function to call

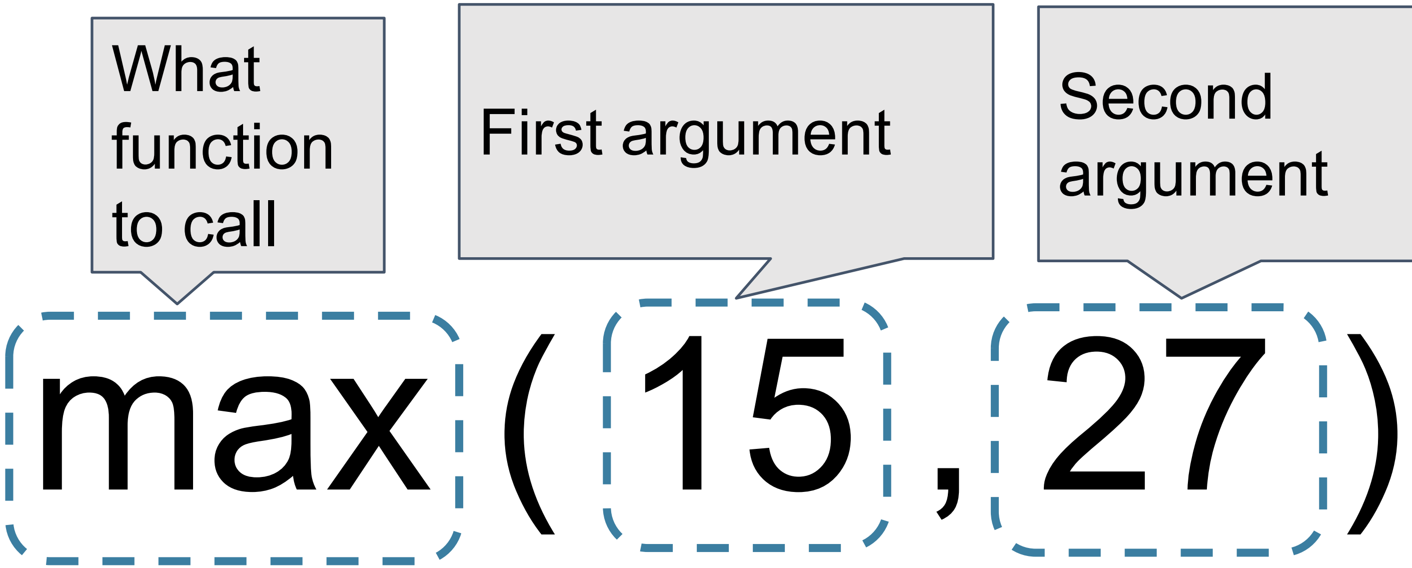
First argument

Second argument

```
max(15,27)
```

Calling Functions

- Here, we using a built-in function “max” that returns the greatest value between some set of numbers.
- We’re saying “find the maximum value of 15 and 27”
  - In this case, the function’s ***return value*** will b e 27.
- If we say “x = max(15,27)” then we do two things:
  - we call the function and we assign “x” to the return value.



What function to call

First argument

Second argument

max (15 , 27)

Calling Methods

- Methods are special functions attached to objects with a dot
  - An object is an actual instance of a data type.

```
title = "Gone west"
title.upper()
```

To deal with a method or values attached to an object, we use the “dot operator” with a syntax of

- object_name.function
- open parentheses
- some arguments
- close parentheses

- Methods can also be chained:

```
title.upper().replace("WEST", "FISHING")
```

Note: Case for strings and variable names matters!

end