Revised: 7/27/2020

Python Fundamentals

Michael C. Hackett

Assistant Professor, Computer Science

Community College of Philadelphia

Lecture Topics

- Data Types
- Variables
  - Assignment, Copying Values
  - Naming Conventions
- Literals
- Comments
- Console Output
- Basic Arithmetic
- Math Module
- Converting ints and floats
- Strings
- Keyboard Input
- Formatted Output

Colors/Fonts

- Variable Names – Brown
- Literals – Blue
- Keywords – Orange
- Operators/Punctuation – Black
- Functions – Purple
- Comments – Gray
- Modules – Pink

Source Code – **Consolas**
Output – Courier New

Data Types

- A ***data type*** specifies the kind of information that data can be.
- It is the *meaning* of the data.
- Data types are used for
  - Specifying the possible values the data can be interpreted as.
  - Specifying what operations can be performed on the data.

Python's Standard Data Types

- Numeric Types
  - int – Integers (Whole numbers)
  - float – Floating-Point numbers (Numbers with decimal places)
- Boolean
  - boolean – Can be true or false.
- Sequences
  - str – Strings (Sequences of characters)
  - list – Sequence of data, the types of which may vary.
  - tuple – Like a list, but values can't be changed/modified.
- Sets
  - set – Unordered collection of data
- Mappings
  - dict – Dictionary (List of data that each have an associated definition)

We will see these data types in future lectures.

There are additional standard data types not shown here.

ints and floats

- The `int` type is used to represent integers, positive or negative.
  - Examples: 57, 0, -23

- The `float` type is used to represent rational numbers, positive or negative.
  - Numbers with a fractional/decimal point.
  - Examples: 64.2, -100.9998

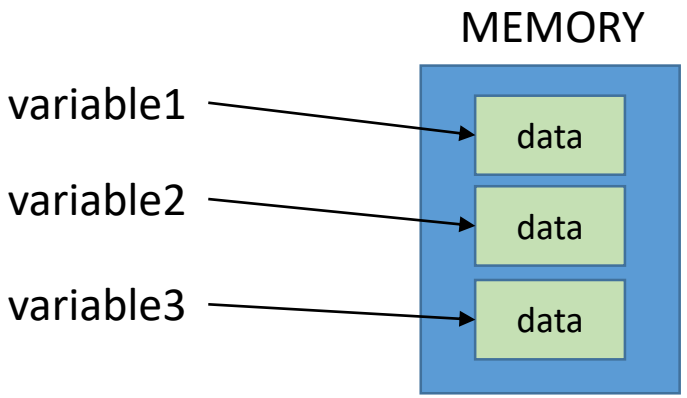
boolean

- The **`boolean`** type can be **`True`** (1) or **`False`** (0).
  - One bit of information.

- Used for decision making.

Variables

- A ***variable*** is an identifier that references the location of an area of memory.



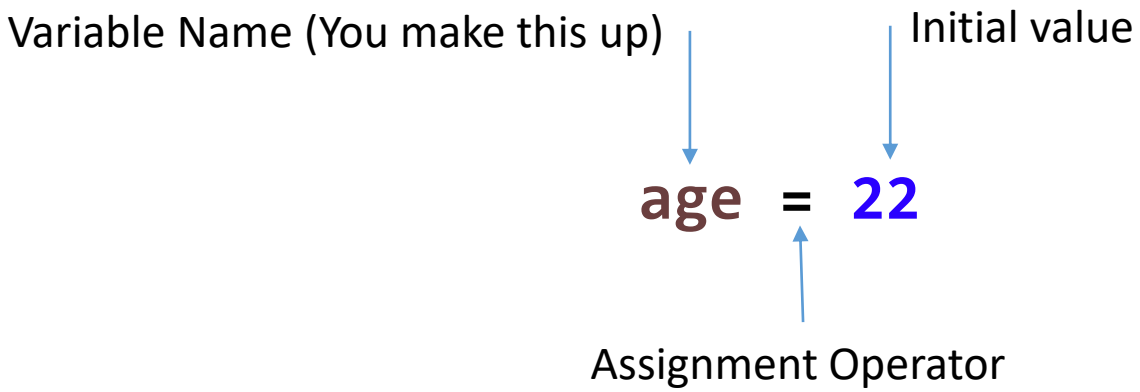
- The type of data a variable contains may vary.

Creating Variables

- Variables are declared and initialized by:
  - Stating the name of the variable, and
  - Giving the variable a value using the assignment operator (=).

- Examples:

```
age = 22
temperature = 80.5
ready = True
```



Assigning a New Value to a Variable

- Replace the existing value of a variable with a new value by using the Assignment Operator.
  - The new data can be of any type.
- Example:

```
temperature = 67.5
temperature = 68.2
temperature = 69
```

Copying Values From One Variable to Another

- Use the Assignment Operator, **=**
- Example:

```
speed = 35.2
speed2 = speed
```

Literals

- A ***literal*** is a source code representation of a fixed value.
  - It is represented without any computation.

- Sometimes referred to as a *hard coded value*.

Literals

- int literals can be expressed in
  - Decimal (Base 10)
  - Octal (Base 8)
  - Hexadecimal (Base 16)
  - Binary (Base 2)
- Decimal Literal (No prefix): `decimal_number = 100`
- Octal Literal (0 or 0o prefix): `octal_number = 0144`
- Hexadecimal Literal (0x prefix): `hex_number = 0x64`
- Binary Literal (0b prefix): `binary_number = 0b1100100`

For the purpose of this course, we will only be using decimal (base 10) literals. It's good to know that other numeric literals exist, though.

Literals

- No prefix or suffix required for float literals.

```
exampleFloat = 255.23
```

Variable Names

- Variable names are case-sensitive.

```
NUMBER = 50
number = 30
```

  - In the lines of code above, NUMBER and number are two separate variables.

Naming Variables

- Names must start with a letter or underscore.
- Names may contain numbers, but **cannot** start with numbers.
- Aside from letters, underscores, and numbers, no other characters may be used.
- Names cannot contain spaces.

```
some_name = 50      Valid.
_some_name = 50     Valid.
1some_name = 50     INVALID.
some_name1 = 50     Valid.
some name = 50      INVALID.
```

Naming Variables

- Variable names in Python are normally all lowercase.

- "Snake-case" is the preferred style (or *convention*) used for variable names in Python.
  - For variable names that are multiple words long, place an underscore between each word.

```
bottles_of_beer_on_the_wall = 99

has_been_deleted = False
```

Strings

- A ***string*** is a standard data type that contains a sequence of characters.

- The sequence of characters in a string can include any number of:
  - Letters
  - Numbers
  - Symbols
  - Spaces

String Literals

- A string literal is a sequence of characters in single quotes (') or double-quotes (")

```
'Hello World!'

"Hello World!"
```

Comments

- Comments are notes programmers leave in the source code to document their code.

- This allows programmers to:
  - Leave notes to themselves.
  - Leave notes to other programmers who may later work on your code.
  - Describe what a section of code does (it may not always be obvious.)

- Alternatively, comments are useful for omitting single or multiple lines when debugging your program.

Comments

- Comments are entirely ignored by the compiler. You can type whatever you want in a comment.

- Single line comments begin with #

```
#Single line comment
```

- Multiple line comments begin with ''' (or """) and end with ''' (or """)

```
''' Everything between quote-quote-quote
    and quote-quote-quote
    will be
    ignored '''
```

Comments

```
i = 10 #Comments can be left after a statement.
```

- Omit an entire line/statement by adding # at the beginning:

```
#j = 15
```

- Omit multiple lines/statements by adding # at the beginning of each, or use multi-line comments:

```
# k = 20
# m = 13
```

```
''' k = 20
    m = 15 '''
```

Comments

- Good comments not only describe *what* a line of source code is doing, but *why* that instruction is important to the program.

```
#Initializes a variable.
number_of_passengers = 10
```

- The documentation/comment above is unhelpful.
  - It's obvious that a variable is being declared and initialized.
  - It does not explain why the variable was initialized to 10.
  - It does not explain why the variable even exists in the first place.
    - Why does the program have this variable? What is it used for?

Comments

- Comments also allow you to omit sections of code without actually deleting them.

- You can later uncomment them, or delete them once you are confident you no longer need the lines any more.
  - If you leave in commented lines of code, you will normally leave another comment explaining why you left them in.

Console Output

- Console Output refers to information or text that a program prints to the screen.

- Python's built-in print function allows us to display console output.
  - A ***function*** is a procedure that, when "*called*", will execute an operation to perform a task.

```
print(value(s) to print)
```

- The parentheses after a function's name contains its argument list.
  - Data passed as an argument to the print function will be printed to the screen as console output.

Console Output

- After printing the supplied information, the print method will advance to the next line.

```
test_score1 = 98
test_score2 = 94
test_score3 = 96
print(test_score1)
print(test_score2)
print()
print(test_score3)
```

```
98
94

96
```

Console Output

- Use a comma separated list to print multiple values at once.

```
test_score1 = 98
print("You scored a", test_score1, "!")
```

```
You scored a 98 !
```

Console Output

- By default, Python's print function places a single space between each value printed.

```
test_score1 = 98
print("You scored a", test_score1, "!")
```

```
You scored a 98 !
```

Console Output

- To change the separator, use a final **sep** argument.
  - The below example causes the print function to use no separator between values.

```
test_score1 = 98
print("You scored a ", test_score1, "!", sep="")
```

```
You scored a 98!
```

Arithmetic Operators

- Addition: **+**
- Subtraction: **-**
- Multiplication: ***
- Float Division: **/**
- Integer Division: **//**
- Remainder Division: **%**
- Exponents: ****

Addition

```
number1 = 6
number2 = 5
sum = number1 + number2
```

- The sum variable is assigned the value 11.

Mixed Type Arithmetic

- Special rules apply when performing arithmetic operations on numbers of different types. For example, adding an int and a float together. *What data type is the result of that arithmetic*?
  - Arithmetic operations performed only on ints result in an int.
  - Arithmetic operations performed only on floats result in an float.
  - Arithmetic operations performed on a combination of ints and floats result in a float.

First Operand	Second Operand	Resulting Type
int	int	int
int, float	float	float

Mixed Integer Operations

```
value1 = 10
value2 = 15
result1 = value1 + value2
```

- The data type of the result1 variable will be int.

```
value3 = 11.7
value4 = 12
result2 = value3 + value4
```

- The data type of the result2 variable will be float.

Mixed Integer Operations

```
value5 = 13.5
value6 = 18.6
result3 = value5 + value6
```

- The data type of the result3 variable will be float.

```
value7 = 21
value8 = 19
value9 = 2.3
result4 = value7 + value8 + value9
```

- The data type of the result4 variable will be float.

Subtraction

```
number1 = 6
number2 = 5
difference = number1 - number2
```

- The difference variable is assigned the value 1.

Multiplication

```
number1 = 6
number2 = 5
product = number1 * number2
```

- The product variable is assigned the value 30.

Float Division

- The float division operator always returns a float result.

```
number1 = 8
number2 = 2
quotient = number1 / number2
```

- The quotient variable is assigned the value 4.0

Float Division (Another Example)

```
number1 = 5
number2 = 2
quotient = number1 / number2
```

- The quotient variable is assigned the value 2.5

Integer Division

- The integer division operator returns a quotient with any fractional portion truncated/dropped.
  - Value returned depends on the data types of the operands.

```
number1 = 8
number2 = 2
quotient = number1 // number2
```

- The quotient variable is assigned the value 4

Integer Division (Another Example)

```
number1 = 10.5
number2 = 2
quotient = number1 // number2
```

- The quotient variable is assigned the value 5.0

Integer Division (Another Example)

```
number1 = 5
number2 = 2
quotient = number1 // number2
```

- The quotient variable is assigned the value 2

Integer Division (Another Example)

- Negative results are rounded away from zero.

```
number1 = -5
number2 = 2
quotient = number1 // number2
```

- The quotient variable is assigned the value -3
  - -5 / 2 = -2.5

Remainder Division

- Finds the remainder of a division.

```
number1 = 11
number2 = 4
remainder = number1 % number2
```

- The remainder variable is assigned the value 3.
- “11 divided by 4 is 2 with a remainder of 3”

Exponents

```
number1 = 2
number2 = 3
result = number1 ** number2
```

- The result variable is assigned the value 8.

Operator Precedence

- PE[MD%][AS] (left to right)
- Multiplication, Integer or Float Division, Mod Division – same priority
- Addition, Subtraction – same priority

```
num1 = 13;
num2 = 5;
num3 = 3;
num4 = 2;

answer = (num1 % num2 * num2) / num3 - num3 ** num4
```

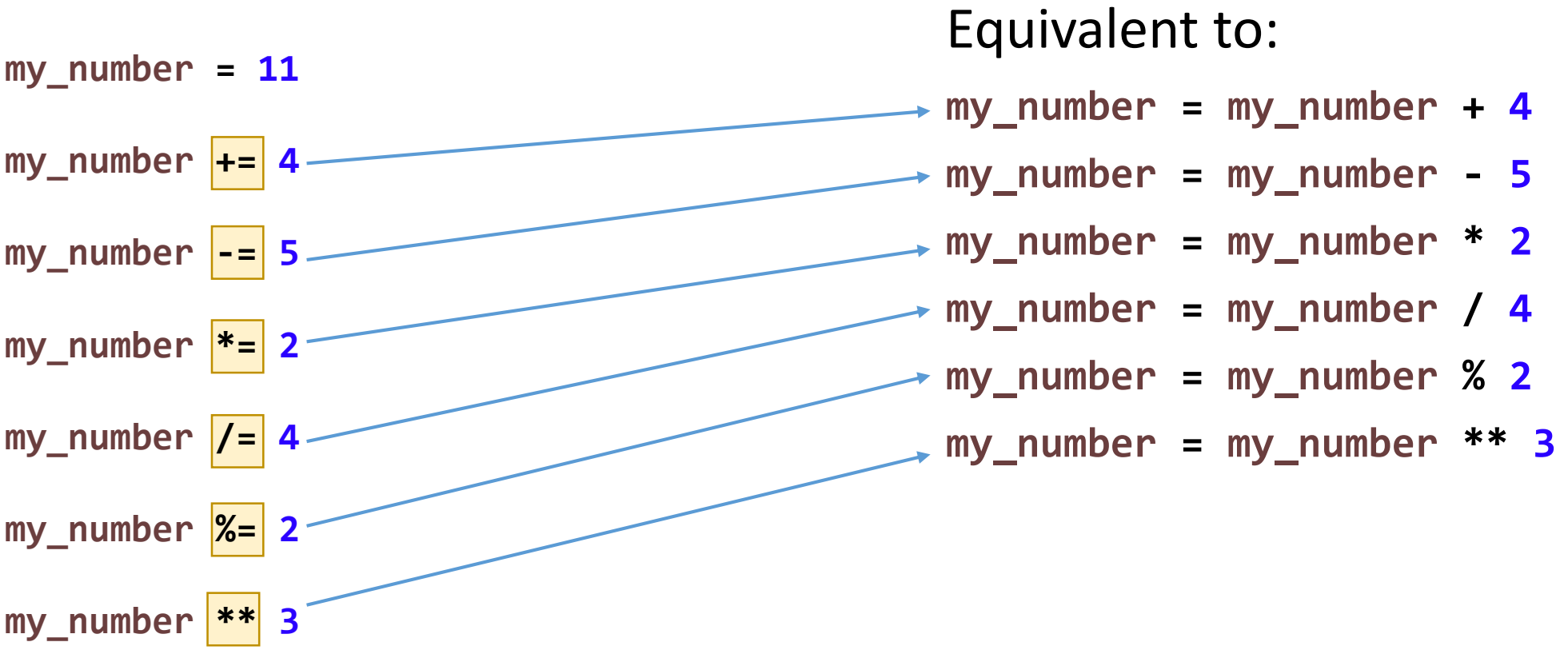
Arithmetic when printing output

- You can preform arithmetic within a print function call.
  - The values referenced by the variables will not be changed.
  - The arithmetic is performed first; the result of the arithmetic is the argument to the print function.

```
total_adults = 20
total_children = 22
print(total_adults + total_children)
print(total_adults)
print(total_children)
```

```
42
20
22
```

Combined Assignment Operators



Rounding floats

- Python's built-in round function will round a float to the nearest whole number.
  - The function's return value is a float type.

```
original_number = 25.6
rounded_number = round(original_number)
print(rounded_number)
```

```
26.0
```

```
original_number = 25.4
rounded_number = round(original_number)
print(rounded_number)
```

```
25.0
```

Python Modules

- A Python module is a file that contains Python code, specifically functions.

- Python comes with many modules, but their functions are not readily available to call upon.
  - Unlike the print or round functions which are always available.

- Modules can be imported into our own programs.
  - Allowing us to use the functions contained within them.

Math module

- The math module provides mathematical functions beyond what is provided by default (like addition and subtraction).
- Common uses are:
  - Rounding values up or down.
  - Square roots
  - Trig functions

- Import the math module using the following statement:

```
import math
```

When/Where to Import Modules

- A module's import statement can appear anywhere in your source code.
- However, the module must be imported before you try to use any of its functionality.
- Most programmers opt to put any and all import statements at the beginning of their source code.

Math module – Rounding Up

- The ceiling (ceil) function rounds a float up.

```
import math
original_number = 15.1
rounded_number = math.ceil(original_number)
print(rounded_number)
```

```
16.0
```

Math module – Rounding Down

- The floor function rounds a float down.

```
import math
original_number = 15.9
rounded_number = math.floor(original_number)
print(rounded_number)
```

```
15.0
```

Math module – Square Roots

- The square root (sqrt) function returns the square root of a number.

```
import math
original_number = 16
square_root = math.sqrt(original_number)
print(square_root)
```

```
4.0
```

Converting ints to floats

- Use Python's built-in float function to convert the value of an int to a float.

```
my_number = 34653
print(my_number)
my_number = float(my_number)
print(my_number)
```

```
34653
34653.0
```

Converting floats to ints

- Use Python's built-in int function to convert the value of a float to an int.
  - Any fractional portion of the float value will be truncated.

```
my_number = 346.87
print(my_number)
my_number = int(my_number)
print(my_number)
```

```
346.87
346
```

String Concatenation

- ***Concatenation*** is the process of joining strings together into one string using the addition operator.
  - This is not the same as *appending*. When you concatenate Strings together, their individual values are not altered.

```
hello = "Hello "
world = "World!"
hello_world = hello + world
print(hello_world)
```

```
Hello World!
```

Note: The values of the string variables hello and world **do not change**.

Concatenating ints/floats with strings

- ints and floats cannot be directly concatenated with strings.

```
first_half = "There are "
days = 31
second_half = " days in January."
sentence = first_half + days + second_half
print(sentence)
```

Concatenating ints/floats with strings

- ints and floats must be converted to a string type using Python's built-in str function.
  - The str function returns the int/float argument in string form.

```
first_half = "There are "
days = 31
second_half = " days in January."
sentence = first_half + str(days) + second_half
print(sentence)
```

```
There are 31 days in January.
```

Note: The days variable is still an int.

Appending to Strings

- ***Appending*** is the process of adding a string value to the end of another string.
  - Unlike concatenation, appending alters values of a string.
  - To append to a string, use the addition combined assignment operator.

```
hello = "Hello "
world = "World!"
hello += world
print(hello)

Hello World!
```

Appending ints/floats onto Strings

- ints and floats must be converted to a string type using Python's built-in str function before the value can be appended.

```
version_number = 2.5
version = "Version "
version += str(version_number)
print(version)
```

```
Version 2.5
```

Note: The version_number variable is still a float.

Converting a String to Uppercase

- A string's upper() function returns a version of itself in uppercase letters.

```
hello = "Hello World!"
print(hello.upper())
print(hello)
```

```
HELLO WORLD!
Hello World!
```

Converting a String to Uppercase

```
hello = "Hello World!"
hello = hello.upper()
print(hello)
```

```
HELLO WORLD!
```

Note that we assign hello.upper() back to the string variable hello. In other words, we are replacing the original value “Hello World!” with “HELLO WORLD!”

Converting a String to Lowercase

- A string's lower() function returns a version of itself in lowercase letters.

```
hello = "HELLO World!"
print(hello.lower())
print(hello)
```

```
hello world!
HELLO World!
```

Converting a String to Lowercase

```
hello = "HELLO World!"
hello = hello.lower()
print(hello)
```

```
hello world!
```

Note that we assign hello.lower() back to the string variable hello. In other words, we are replacing the original value “HELLO World!” with “hello world!”

Converting Strings to numbers

```
ten = "10"
result = ten + 15
```

- The above code will not work. You cannot perform arithmetic with strings, even if the string's characters are numbers.

- Numeric strings must be converted to int or float form before you can use them as a numeric type.

Converting Strings to ints

- We can use Python's built-in int function to get the numeric value of a string as an int.

```
ten = "10"
result = int(ten) + 15
print("The result is", result)
```

```
The result is 25
```

Note: The ten variable is still a string.

Converting Strings to floats

- We can use Python's built-in float function to get the numeric value of a string as an float.

```
ten = "10.57"
result = float(ten) + 15
print("The result is", result)
```

```
The result is 25.57
```

Note: The ten variable is still a string.

ValueError Exception

- A **ValueError** exception is an error that will occur when you try to convert a string that isn't a number into a numeric type.

```
letters = "abcd"
to_number = int(letters)
```

```
>>> letters = "abcd"
>>> toNumber = int(letters)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'abcd'
>>>
```

Escape Sequences

- \n – Line Feed
- \’ – Single Quote
- \” – Double Quote
- \\ – Backslash

- There are more, but we will only be working with these few.

Escape Sequences - \n

- \n inserts a line feed (or starts a new line)

```
print("Hello \nWorld")
```

```
Hello
World
```

Escape Sequences - \'

- Inserts a single quote character
  - Without this, the interpreter will interpret the ' as the start/end of a String literal.

```
print('\'Hello\' World')
```

```
'Hello' World
```

Escape Sequences - \"

- Inserts a double quote character
  - Without this, the interpreter will interpret the " as the start/end of a String literal.

```
print("\"Hello\" World")
```

```
"Hello" World
```

Escape Sequences - \\

- Inserts a backslash character.
  - The single backslash indicates the start of an escape sequence to the interpreter.
  - So, the backslash character itself needs to be escaped.

```
print("Hello \\ World")
```

```
Hello \ World
```

Getting Keyboard Input

- Python's built-in input function allows us to prompt the user to enter input.
- One argument: A string that serves as the prompt.

```
name = input("Enter your name: ")
print("Nice to meet you ", name, "!", sep="")
```

```
Enter your name: John
Nice to meet you John!
```

Getting Keyboard Input

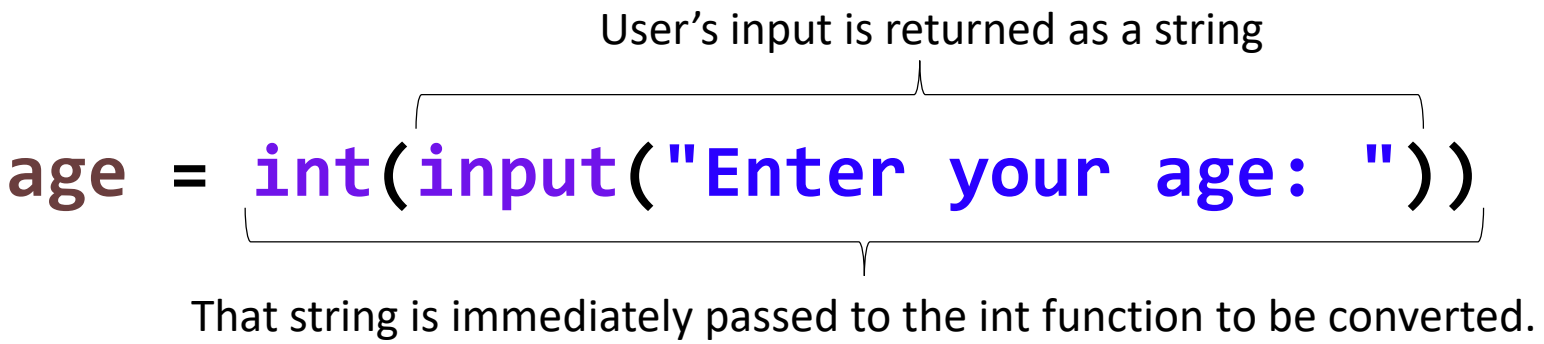
```
name = input("Enter your name: ")
age = input("Enter your age: ")
print("Nice to meet you ", name, "!", sep="")
print("You are ", age, " years old.", sep="")
```

```
Enter your name: John
Enter your age: 45
Nice to meet you John!
You are 45 years old.
```

Getting Keyboard Input

- The input function always returns the user's input as a string.
  - Even if they entered a number.

- If you plan on using the value the user enters as a numeric type, you will need to convert the value to an int or float.



Formatting Numbers

- Python's built-in format function allows us to include commas, decimal rounding and scientific notation for printing numbers as console output.
  - The return value of the format function is a string.

- The format function accepts two arguments:
  - The numeric value to format (int or float.)
  - A string that specifies the format.

Formatting Numbers

- The format specifier for int values is the letter **d**.

```
age = 75
formatted_age = format(age, "d")
print(formatted_age)
```

```
75
```

- The format specifier for float values is the letter **f**.

```
temp = 98.6
formatted_temp = format(temp, "f")
print(formatted_temp)
```

```
98.600000
```

Formatting Numbers

- The format specifier for int/float values for scientific notation is the letter **e**.

```
distance = 1037438.675
formatted_dist = format(distance, "e")
print(formatted_dist)                    1.037439e+06
```

Flags

- Flags are optional and specify how the number is to be formatted.
- Inserted before the specifier's character.
  - Comma Flag (works for ints and floats)

```
amount = 2500000
formatted_amt = format(amount, ",d")
print("The amount is $", formatted_amt, sep="")
```

```
The amount is $2,500,000
```

Flags

- Flags are optional and specify how the number is to be formatted.
- Inserted before the specifier character.
  - Decimal Flag (works for ints and floats)

```
seconds = 74.3647
formatted_sec = format(seconds, ".3f")
print(formatted_sec, "seconds.")
```

```
74.365 seconds.
```

Flags

- Comma and Decimal Flag (works for ints and floats)
  - Comma flag must come first.

```
value = 5434.528
formatted_value = format(value, ",.2f")
print("The value is", formatted_value)
```

```
The value is 5,434.53
```

Flags

- Scientific Notation
  - Decimal Flag (works for ints and floats)

```
value = 5434.528
formatted_value = format(value, ".2e")
print("The value is", formatted_value)
```

```
The value is 5.43e+03
```