Revised: 1/21/2019

Variables, Data Types, and Data Streams

"Computers are good at following instructions, but not at reading your mind."

-Donald Knuth

Michael C. Hackett
Computer Science Department

Community College of Philadelphia

Lecture Topics

- Programming Language Concepts
- Data Types and Literals
- Variables
    - Assignment
    - Copying
    - Swapping
- Standard Streams
    - Standard Output
    - Standard Input
    - Standard Error
- Comments

Colors/Fonts

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

Source Code – **Consolas**

Output – `Courier New`

Basics of a Programming Language

- Programming Languages (modern, high-level languages, at least) incorporate the following concepts:
  - Keywords
  - Operators
  - Punctuation
  - Syntax
  - Variables

Keywords

- A ***keyword*** (or reserved word) is word that has special meaning to a programming language.
  - The word is *reserved* from being used in other contexts within programs written in the language.
  - Keywords are typically used in a language for performing some specific process.
- For example, in many languages the word “if” is a reserved word.
  - The if keyword begins a special statement that allows a program to make a decision.
  - **if** *this* then do *that*
- Many different languages utilize the same keywords.

Operators

- An ***operator*** is (usually) a symbol that performs an operation on one or more operands(values/data).

- In the mathematical expression 1 + 2, the plus sign is an operator that adds the two operands together.
  - In this example:
    - **1** and **2** are operands
    - **+** is the operator
    - **Addition** is the overall operation performed by the operator.

- Many languages use the same operators for performing common operations, like arithmetic and comparisons.

- In some cases, keywords can take the form of an operator.

Punctuation

- ***Punctuation*** is characters or symbols used when writing statements in a programming language.
    - A *statement* is like a sentence or an instruction in a programming language.

- Consider the sentence *I went to the park, the mall, and the college.*
    - We used punctuation for listing multiple places (commas) and a period to end the sentence.
    - Programming languages will use characters in similar ways.
        - For example, commas are often used in programming languages when specifying a list of values.

- Punctuation varies among different languages.
    - Some languages, like Java and C require ending statements with semicolons.
    - Languages like Python do not require punctuation at the end of statements.

Syntax

- ***Syntax*** is the language's rules for how keywords, operators, punctuation, and identifiers must be arranged in statements.

- The rules for how statements are written are paramount.
    - It ensures the statements and instructions of a program are correctly executed.

- "Tall, he is."
    - We can kind-of understand what this English statement is saying.
    - A computer can't "guess" our intentions when we give it instructions.
        - A statement is syntactically correct, or it is not. There can be no ambiguity.

Syntax

- A language’s syntax is usually the most notable difference among different programming languages.
  - How languages accomplish tasks is comparable, but how we write those statements to accomplish the task usually differs.

- Some languages have comparable syntax.
  - Many languages are derived from or inspired by other languages.
  - Java and C++ have comparable syntax as they are both heavily based on the C programming language.
  - Python and Java have some similarities, but overall have many differences in syntax.

Data Types

- A ***data type*** specifies the kind of information that data can be.

- It is the *meaning* of the data.
  - The type identifies how the data can be used.

- Data types are used for
  - Specifying the possible values the data can be interpreted as.
  - Specifying what operations can be performed on the data.

Data Types

01101101

- All computerized information is represented in binary digits (bits) consisting of 1’s and 0’s.

- The binary digits above could represent…
  - The decimal number 109, or
  - The letter "m" in ASCII character encoding

- Data types are used in a programming language to ensure the binary information in memory is interpreted correctly.

Data Types

- All languages have low-level data types for use.
  - Python- Standard Data Types
  - Java- Primitive Data Types

- These low-level types typically share similarities across different languages.

- These low-level data types are the building blocks for more high-level, complex types.

Numeric Types

- Programming languages generally have two types for numeric values.
  - Integers
  - Floating Point Numbers (“Floats”)

- Some languages, like Java, have multiple types for integers and floating point numbers.

- Python only has one for each.

Integers

- An ***integer*** is a whole number.
    - 26
    - 0
    - -5

- Integers do not have fractional portions.
    - 45.7 is not an integer.

Floating Point Numbers

- A floating point number is used to represent a rational number, or numbers with fractional amounts.
  - 56.7
  - 0.86
  - 4.019999
  - -31.5

- The binary information that makes up a floating point number ("float") is organized in a special way.

Boolean Types

- A ***boolean*** value can be either true or false.
    - Often represented using 1 bit of information.
        - 0 being false and 1 being true.

    - Depending on the system and how memory is allocated, a boolean value may be longer than one bit in length.

Strings

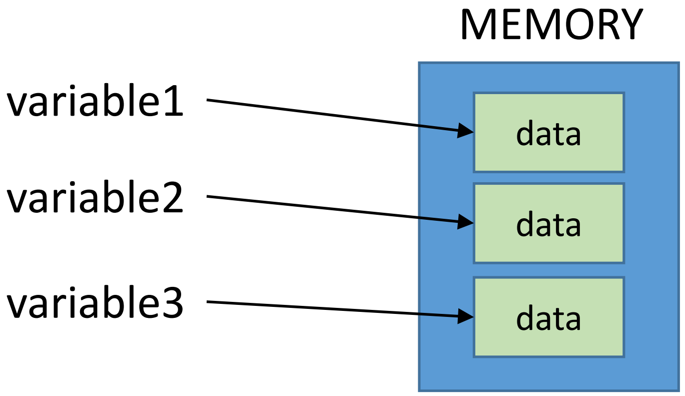
- A ***string*** is a data type that contains a sequence of characters.

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

Variables

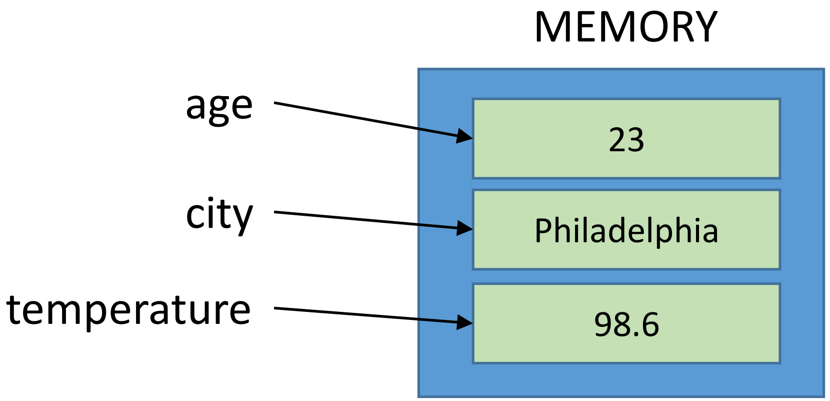
- A ***variable*** is a type of identifier that represents a reference to a location in memory where data is stored.

- Like the name suggests, the data referenced by a variable may vary.
  - New values/data can be assigned to the memory location the variable references.



Variables

- Variable names are programmer defined.
    - We choose variable names based on the data they represent in our programs.



Creating Variables

- As previously stated, variables are identifiers that reference a location in memory.
  - The name of a variable is decided by the programmer.

- To assign data to the memory location referenced by a variable, we use the assignment operator **=**

- The syntax is nearly always ***variable = value*** regardless of the language.

- Based on that syntax, a compiler or interpreter understands…
  - The identifier before the assignment operator is the variable.
  - The value after the assignment operator is the data we want to store to the location in memory the variable references.

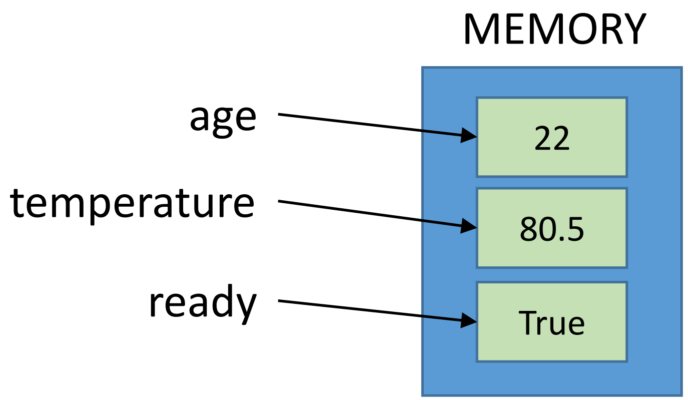
Creating Variables

- A variable is ***declared*** at the time of its first occurrence.
  - If this was the first time we used a variable named age (below), memory will be allocated for the variable to reference. The value will then be stored at that memory location.

Variable

Value/Data

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



MEMORY

age

22

temperature

80.5

ready

True

Variable Names

- Variable names are case sensitive.
    - Pay close attention to any capital letters and spelling.
    - In the lines below, NUMBER and number are two separate variables.

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

- A keyword cannot be used as a variable's name.

Variable Names

- 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
```

Literals

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

- Sometimes referred to as *hard coded values*.

Numeric 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.

Numeric Literals

- No prefix or suffix required for float literals in Python.

```
example_float = 255.23
```

Boolean Literals

- Literal Boolean values are either True or False.
  - Both are keywords.
  - Uppercase T and F.

```
example_bool1 = True
example_bool2 = False
```

String Literals

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

```
example_str1 = 'Hello World!'

example_str2 = "Hello World!"
```

- In some languages, only double-quotes are permitted for string literals.

Variables and Data Types

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

- A variable’s data type in Python is *dynamically typed*.
    - Since we assigned 22 to the memory location referenced by the variable age, Python infers that value to be an int.
        - This kind of dynamic typing is called *duck typing*. (“If it walks like a duck and quacks like a duck, it must be a duck.”)

- Other languages, like Java, are *statically typed*.
    - Static typing is when variables are restricted to referencing a specific data type.
    - For example, a variable’s type would be declared and only allow the variable to reference memory locations containing values of that type.

Strongly Typed vs Loosely Typed Languages

- Programming languages can be categorized as strong typed or loose typed.
  - No universally agreed upon definition of either.
  - This falls under the broader topic of *type safety*.

- A **strongly typed language** is a language that performs type checks.
  - Either at compile time or at run time.

- A **loosely** (or *weakly*) **typed language** is one that does not perform type checks.

Strongly Typed vs Loosely Typed Languages

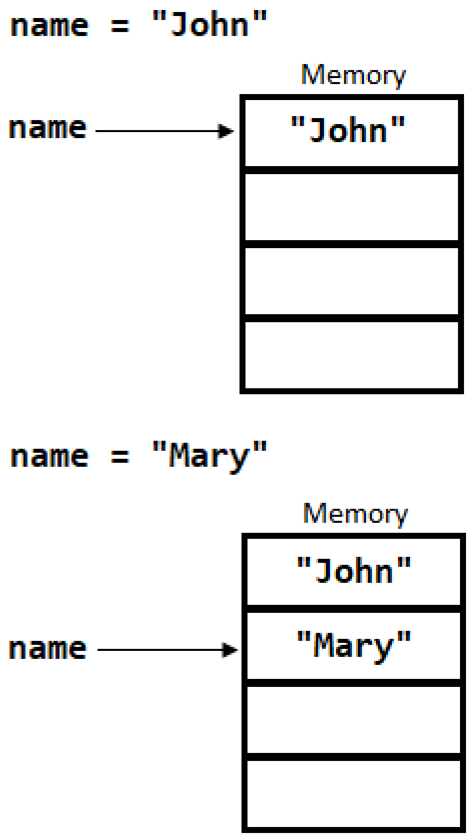
- Java is considered strongly typed.
  - Data types must be declared (*static typing*) and Java code will not compile if there are type mismatches.
  - Type checks are performed at compile time.

- Python is also considered strongly typed.
  - Does not have static typing like Java.
  - Type checks are performed at run time.

Reassignment

- To reassign a new reference to a variable, the assignment operator is again used to associate the new reference with the existing variable.

```
name = "John"
name = "Mary"
```



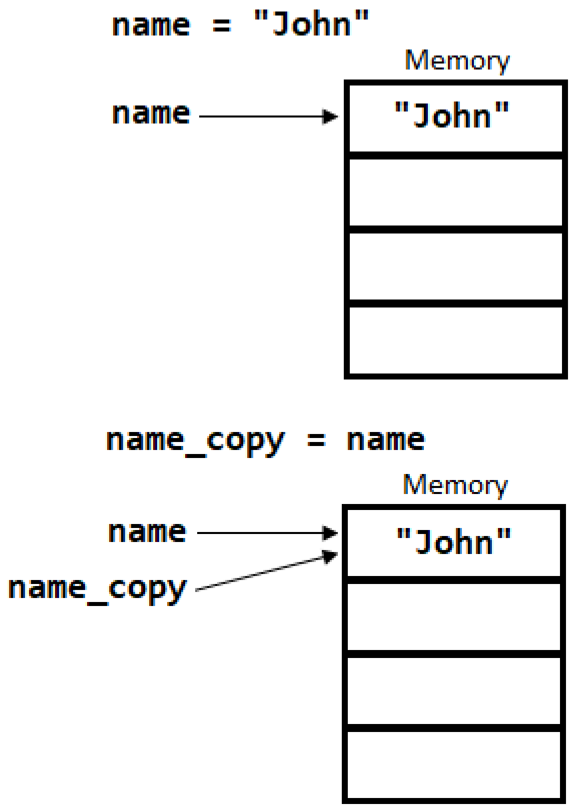
```
name = "John"
```

```
name = "Mary"
```

Copying

- Copying a variable's reference is similar to the process of reassignment.
  - The end result is two (or possibly more) variables referencing the same data.

```
name = "John"
name_copy = name
```



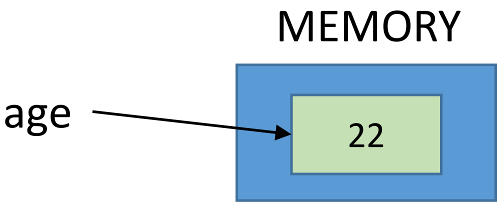
Immutable vs Mutable Data

- ***Immutable*** data is data in memory that cannot be changed.
- ***Mutable*** data is data in memory that can be changed.
- Depending on the programming language, some data types are immutable while others are mutable.

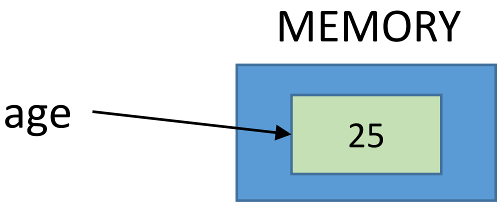
Mutable vs Immutable Data

- Mutable

```
age = 22
```

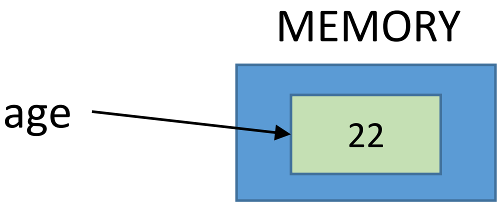


```
age = 25
```

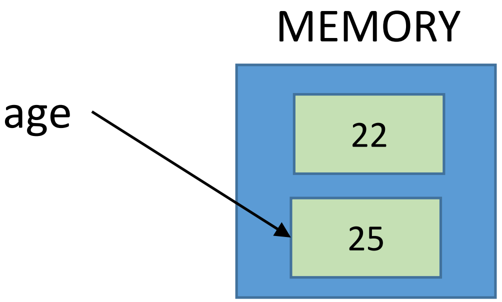


- Immutable

```
age = 22
```



```
age = 25
```



Swapping

- The following algorithm can be used to swap the references of two variables.
    - A third variable is needed to temporarily hold a reference.

1. Copy the first variable to the temporary variable
2. Copy the second variable to the first variable
3. Copy the temporary variable to the second variable

Swapping

```
distance1 = 91.5
distance2 = 88.2
```

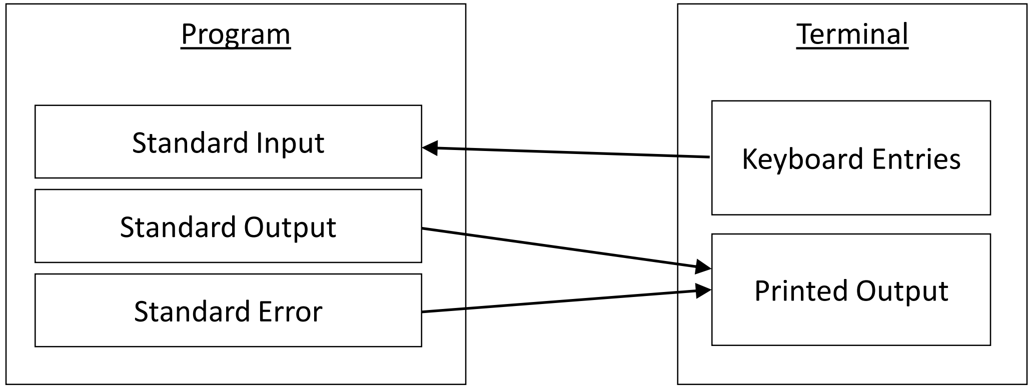
```
temp = distance1
distance1 = distance2
distance2 = temp
```

Standard Streams

- A computer program uses data streams for handling incoming and outgoing data.

- Three standard streams
  - Standard Output (“stdout”, “standard out”)
  - Standard Input (“stdin”, “standard in”)
  - Standard Error (“stderr”, “standard error”)

Standard Streams



Standard Output

- ***Standard Output*** refers to the data stream used to print information/text to a terminal.
  - The term *console* or *terminal* is used to describe a text-only interface.

- Python's built-in print function allows us to display console output.

```
print(values 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 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)                  98
print(test_score2)                  94
print()
print(test_score3)                  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!
```

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
```

Standard Input

- ***Standard Input*** refers to the data stream used to receive keyboard entries.

- Python's built-in input function allows us to
  - Prompt, or ask, the user to enter data using the keyboard
  - Store the user's input to memory

```
variable = input(string to print to the user)
```

- The input function accepts only one string argument.

Keyboard Input

- The value returned by the input function will always be a string.
  - If you intend to use the user’s input as a numeric type, it will need to be converted (covered in a later lecture).

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

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

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.
```

Standard Error

- ***Standard Error*** refers to the data stream used to print or record errors.

- Standard Error and Standard Output are both concerned with information coming out of the program.
  - Having two streams lets us differentiate between normal output and error messages.

- No special function is used.

Standard Error

- Entering the below statement into the Python interpreter will cause a syntax error.

```
x = 7k
```

- Error Message:

```
File "<stdin>", line 1
    x = 7k
         ^

SyntaxError: invalid syntax
```

Syntax Error

- A ***syntax error*** is caused by a statement that an interpreter or compiler cannot understand.
  - The statement breaks the rules for how statements must be written in that programming language.

- In this case, the interpreter doesn't understand what "7k" means.

```
File "<stdin>", line 1
    x = 7k
         ^

SyntaxError: invalid syntax
```

Name Error

- A ***name error*** is caused by a statement that refers to an identifier that does not exist.
  - Usually caused by a misspelled variable or function name.
  - Variables and function names are case sensitive.

```
temp = 98.6
temp_copy = tmep
```

```
File "<stdin>", line 1

NameError: name 'tmep' does not exist
```

```
temp = 98.6
prnit(tmep)
```

```
File "<stdin>", line 1

NameError: name 'prnit' does not exist
```

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.

- Inline (or single-line) comments in Python begin with #

```
#Single line comment
```

- Multi-line comments in Python 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.

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

- 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.