Computer Programming I

Introduction to Python

What is an algorithm?

- An algorithm is a description of a process in a step-by-step manner.
  - List of the steps needed to solve a problem
- Analogy: a recipe

What is a program?

- A computer program is a series of instructions that tell a computer how to perform a task
  - Executing one instruction at a time
  - Written in a programming language
- A program is the implementation of an algorithm.

Instruction Types

- Basic instruction types in a program are:
  - ***Input***: A program gets data
    - from a file, keyboard, touchscreen, network, etc.
  - ***Process***: A program performs computations on that data
  - ***Output***: A program puts that data somewhere,
    - a file, screen, network, etc.
- All programs follow the general pattern:
  - Input → Process → Output

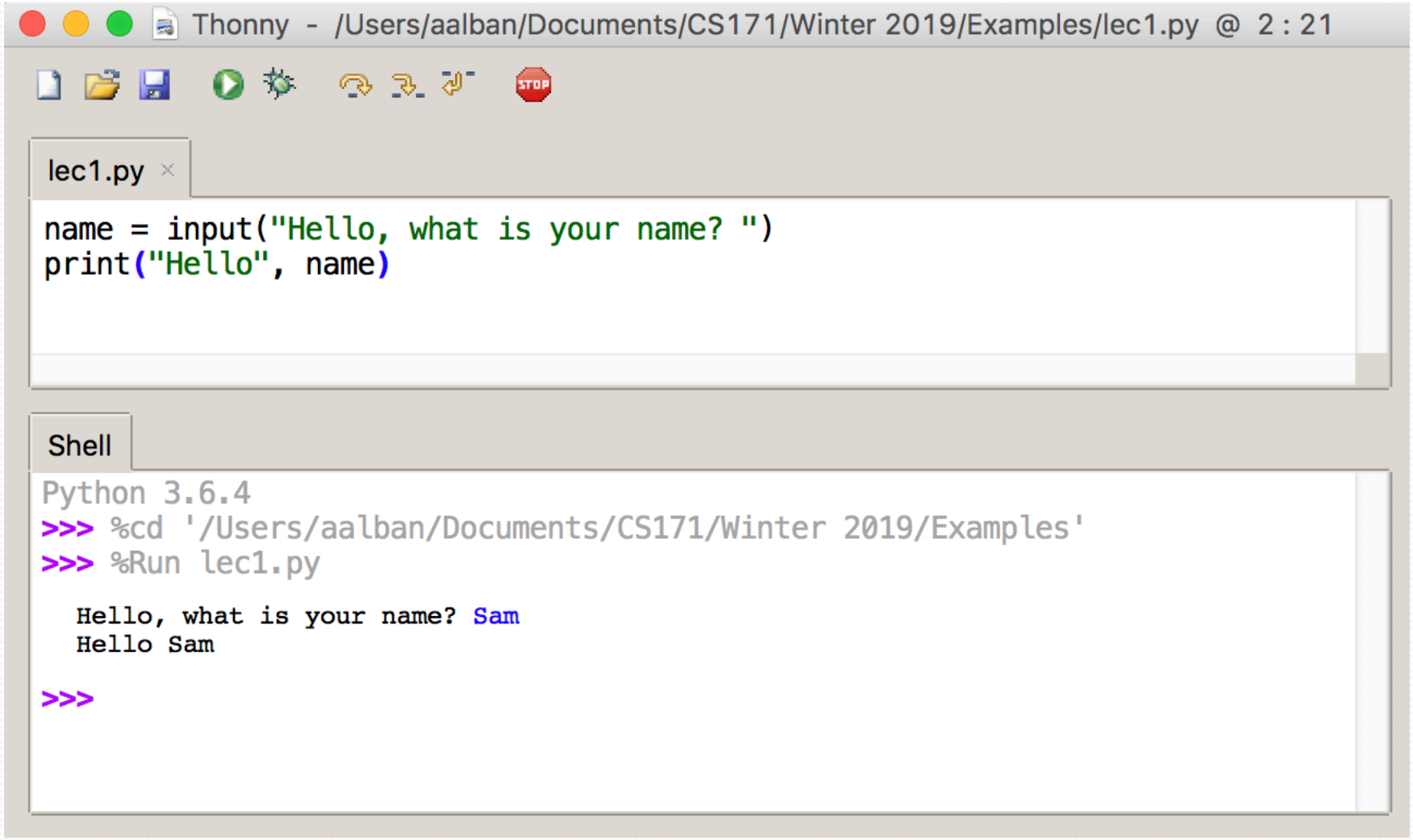
Input/Process/Output Example

- Sending an email:
  - ***Input***: The body of the email and address to send to
  - ***Process***: Connect to the Internet and transfer the email to the recipient
  - ***Output***: The recipient reads your email
- Most programs repeatedly perform these three actions.
- Playing a video game:
  1. ***Input***: Check what buttons the player hit
  2. ***Process***: Update the game (move characters on screen, kill enemies, etc.)
  3. ***Output***: Draw another frame of video for the monitor
  4. Return to step 1

IDE

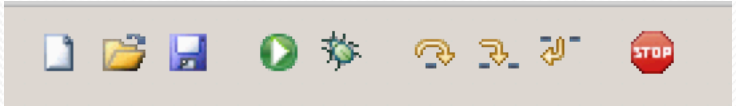
- IDE: Integrated Development Environment
- An IDE is a program that allows programmers develop software
- IDEs include tools for:
  - Writing code
  - Debugging code
  - Executing code
- Thonny is the IDE we will be using in this course
- Thonny can be obtained for free at: http://thonny.org/

Thonny


Thonny - /Users/aalban/Documents/CS171/Winter 2019/Examples/lec1.py @ 2 : 21
STOP
lec1.py
name = input("Hello, what is your name? ")
print("Hello", name)
Shell
Python 3.6.4
>>> %cd '/Users/aalban/Documents/CS171/Winter 2019/Examples'
>>> %Run lec1.py
Hello, what is your name? Sam
Hello Sam
>>>

Thonny Basics

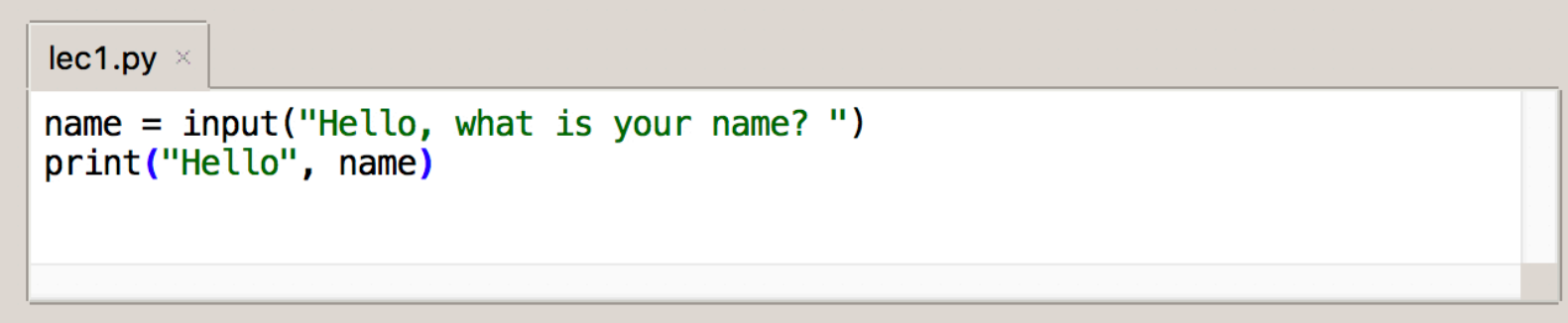
- The top row of buttons has the main features:



- Features:
  - Create Program
  - Open Program
  - Save Program
  - Run Program
  - Debug Program
  - The next 3 buttons move the debugger through the code
  - Stop forces the program to quit

Thonny Basics

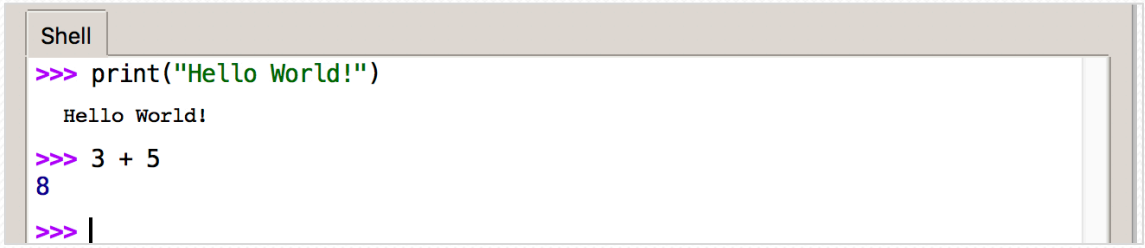
- The top window is a text editor where Python code is written.



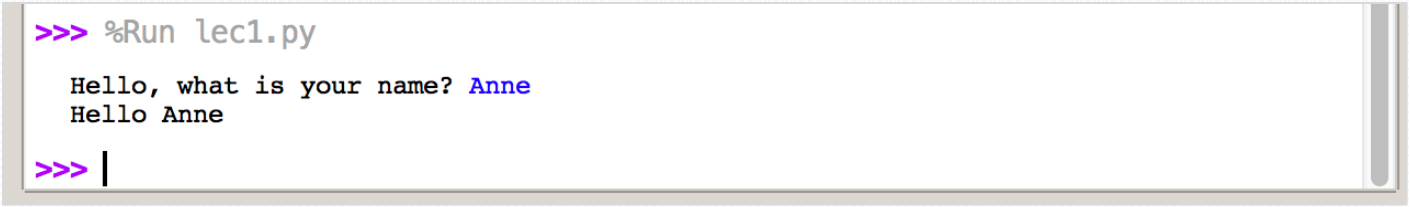
- Save the file with the `.py` extension.
- This file is called the source code for the program.

Thonny Basics

- The bottom window is the interactive interpreter
- You can issue all sort of commands in the interactive interpreter



- You can execute your source code in the interpreter
  - The interpreter executes the instructions written in your source code.



Source Code

- **Code** is the plain text representation of the program.
- A **line** of code is a single row of text.
- A **statement** is an instruction in the code.
- A **program** is just a collection of statements executed in order.

Variables

- Variable: a storage location in memory
- Used to store data used in our programs as well as results computed by our programs
- In Python we create variables by using the assignment operation using the = symbol (assignment operator)
- Examples:

```
year = 2020
course = 'CS 171'
```

- Values of variables can change (vary) during the execution of a program

The assignment operator

- **NOTE**: the variable receiving the value must appear on the left side of the =
- This will NOT work:

```
// ERROR!
20 = age
```

- Please note that in programming = does not have the same meaning as in math.
- The = should be interpreted as: "*compute the value on the right, and then assign that value into the variable on the left*"

Variable Naming Rules

- You may use letters (a – z or A – Z), digits (0 – 9), or underscore (_) characters.
- The first character must be a letter or an underscore
- The name cannot contain spaces or any other special characters
- The name cannot be a reserved word (keyword)
- Always choose a meaningful name that represents the value stored in the variable
- Python is case sensitive
  - upper and lower case letters are not the same ($A \neq a$)
- Names are also known as *identifiers*.

Naming Conventions

- Sometimes to make a name more descriptive you will need to use more than one word
  - Example: you want a variable for the number of students
- There are two style conventions for your function and variable names. You can use either one, but be consistent through the program:
  - Pothole case: lowercase letters with words separated by underscores.
    - Example: number_of_students
  - Camel case: no spaces in between words, words after the first one start with uppercase letter.
    - Example: numberOfStudents

Basic Output

- Output to a screen is done with the built-in function **`print()`**
- We can print text into the screen by enclosing it in quotes.
  - Text enclosed in quotes is known as a string literal.
  - A string literal can be surrounded by matching single or double quotes (`' '` `" "`)
  - Text in string literals may have letters, numbers, spaces, or symbols
- Example:

```
print ("Hello!")
print ('Welcome to CS 171!')
```

Basic Output

- Each print statement will output on a new line.
- If we want to keep the next piece output in the same line, we can add the command **`end =' '`** inside the **`print`** function.
- Example:

```
print ("Hello!", end =' ')
print ('Welcome to CS 171!')
```

Will produce:

```
Hello! Welcome to CS 171!
```

Basic Output

- By being able to use single and double quotes Python allows us to nest quotes
  - Remember to begin and end the string literal with the same type of quote
- Example:

```
print("Don't count the days, make the days count")
```

Basic Output

- The **`print`** function can be used to display the value of a variable
- Example:

```
>>> name = "Joe"
>>> age = 20
>>> print(name)
Joe
>>> print(age)
20
```

- You can give **`print`** multiple values and it will print them with spaces.

```
>>> print("Your name is", name, "and you are", age)
Your name is Joe and you are 20
```

Basic input

- To get input from the user we use the `input()` built-in function
- `input()` causes the interpreter to wait until the user has entered some text and has pushed the return key.
- The input obtained by `input()` is any text that a user typed
  - numbers, letters, or special characters
- We need to use variables in order to save the input given by the user.

Input Example

```
>>> print ("Enter your name: ")
Enter your name:
>>> name = input()
Joe
>>> print ("Enter your age: ")
Enter your age:
>>> age = input()
20
```

Input and prompt

- Adding a string inside the parenthesis of **`input()`** displays a prompt to the user
- Example:

```
>>> name = input("Enter your name: ")
Enter your name: Joe
```

Input

- The input function always returns a string type (text)

```
>>> age = input("Enter your age: ")
Enter your age:
20
>>> print(age)
20 ← This is text, not a numeric value
```

- In order to read a numeric value the **`int()`** function can be used in combination with the **`input()`** function

```
>>> age = int(input("Enter your age: "))
Enter your age:
20 ← This is a numeric value
```

Input

- If we want to read numbers that have a decimal part, we need to use the built-in function **`float()`**
- Example:

```
>>> gpa = float(input("Enter your gpa: "))
Enter your gpa: 3.75
```

Debugging your programs

- A bug is an error in your program
- There are three kinds of errors
  - Syntax errors
  - Logic or semantic errors
  - Runtime errors

Syntax Error

- Syntax error is a mistake such as a:
  - Misspelled word
  - Missing punctuation character
  - Incorrect use of an operator
- A syntax error occurs when a statement in the program violates the rules of the programming language
- A syntax error must be fixed before the program can be executed
- The interpreter will generate a message when encountering a syntax error.

Logic Errors

- A logic error causes the program to operate incorrectly, but not to fail
  - You can still run the program, but you will get erroneous or unintended results.
  - For example, using multiplication where you meant to use addition
- The interpreter does not find these errors
- Since the program will still run, the programmer must be careful examining the results of the program to detect if there is a mistake in the logic of it

Runtime errors

- An error that occurs during the execution of a program.
- A program crash is the most noticeable type of runtime error, since the program unexpectedly quits while running.
- Common examples:
  - dividing by zero
  - referencing missing files
  - calling invalid functions
  - not handling certain input correctly

Comments

- Brief notes in the source code
- Used to document parts of the program
- Comments are ignored by the interpreter
- In Python we start comments with the # sign
- Comments can start any where in a line, not just the beginning of the line
- Example:

```
#this is an example of a comment
```

Good use of comments

- All programs should have a header comment
  - At the top of the file, explaining:
    - Purpose of the program
    - The name of the programmer and contact information
    - Last time the program was modified
- Use comments to label different sections of the program
- Use comments to explain complex lines of code

Blank Lines and whitespaces

- Whitespace: blank spaces between items within a statement.
- Blank Line: blank line between statements.
- Used in the code to create a sense of visual organization
  - Separate different sections of the program
  - Make blocks of code stand out
- Make the code easier to read

Arithmetic Expressions

- The simplest type of processing it to evaluate mathematical expressions.

SYMBOL	OPERATION	Description
+	Addition	Adds two numbers
-	Subtraction	Subtracts one number from another
*	Multiplication	Multiplies two numbers
/	Division	Divides left hand operand by right hand operand
//	Floor Division	Rounds down the result of a division to the closest whole number value.
%	Remainder, or modulo	Divides left hand operand by right hand operand and returns remainder

Programmatic Thinking

- To write good programs, it is important to think programmatically.
- Break problems down into small general steps
- How to make change?
  1. Change = Amount Given - Price
  2. How many Dollars are needed to return Change?
  3. How many Quarters are needed to return Change?
  4. How many Dimes are needed to return Change?
  5. How many Nickels are needed to return Change?
  6. How many Pennies are needed to return Change?

Simple Program

- **Goal:** Calculate the final price of an item
  - Add Tax and Discount to an item's price.
- **Input:** Ask the Price
- **Process:** Compute Tax and discount. Update Price
- **Output:** Display the new price

Source Code

```
tax = 0.08       #8% sales tax
discount = 0.1   #item is 10% off

# get input from the user
price = float(input("Enter item's price: "))

# apply price deduction
deduction = price * discount
price = price - deduction

# apply sales tax
increase = price * tax
price = price + increase

# display final price
print("Final Price for the item:" , price)
```

Resulting Output

```
Enter item's price: 9.99
Final Price for the item: 9.71028
```

Functions

- Function: a named piece of code that performs a specific task.
  - Examples: **`int()`**, **`print()`**
- Function call: invoking the function name causes the function to execute.

```
>>> print('Hello')  #function call
Hello
```

- Arguments and Parameters: values given as input to the function.

```
print('Hello') #'Hello' is the input value given to
               # the print function
```

Functions

- Return value: Some functions will return a result after they have completed the task.
  - Example:

```
>>> intFive = int('5') #input is the character '5'
>>> print(intFive) #returned value is the number 5
5         #returned value – an integer
```

Functions

- We can create our own functions.
- Follow this format:

```
def functionName(inputName1, inputName2, . . . ) :
    # body of the function goes here
    # for example calculations
    # return value from function
```

- Example:

```
def addValues(number1, number2) :
    result = number1 + number2
    return result
```

Final price with Functions

```
# function that calculates final price of an item
def calculateFinalPrice(price, tax, discount):
        # apply price deduction
        deduction = price * discount
        price = price - deduction

        # apply sales tax
        increase = price * tax
        price = price + increase

        # return final price
        return(price)
```

Final price with Functions

```
tax = 0.08       #8% sales tax
discount = 0.1   #item is 10% off

# get input from the user
price = float(input("Enter item's price: "))

# call function
finalPrice = calculateFinalPrice(price, tax, discount)

# display final price
print("Final Price for the item:" , finalPrice)
```