NEW PERSPECTIVES

Unit E
Step-by-Step: Programming with Python

Computer Concepts 2016
ENHANCED EDITION

Unit Contents

- Section A: "Hello World!"–Python Style
- Section B: The Wacky Word Game
- Section C: Build Your Own Calculator
- Section D: Ask The Fortune Teller
- Section E: Dogs and Cats

Section A: "Hello World!–Python Style

- Programming Basics
- Introduction to Python
- Let's Start Coding
- Working with Strings
- Using Python Keywords

Programming Basics

- A **computer program** is a set of step-by-step instructions that tells the computer what to do.
- **Computer programming**, otherwise known as **programming**, is the process of writing the instructions, referred to as **code**, that tell the computer what to do.
- Instruction code for a computer program is based on an **algorithm**, which is a sequence of steps for solving a problem or performing a task.

Introduction to Python

- Python is a programming language used to communicate with a computer.
- Other types of programming languages include:
  - C
  - C++
  - Java
  - JavaScript

Introduction to Python

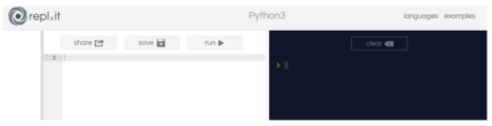
- Python is excellent for beginners, yet outstanding for experts.
- To work in Python you'll need the following:
  - A **Code Editor** – a place to enter source code for a program.
  - A **Debugger** – a computer program used to find errors.
  - An **Interpreter** – a program that translates code into machine language.
  - A **Compiler** – a program that translates code to a machine language before sending it to the computer; these can be Web apps.

1

Introduction to Python

- An IDE (integrated development environment) provides an editor, debugger, and interpreter.
- The programs in Unit E use the online IDE **repl.it**.

FIGURE E-1: THE IDE FOR PYTHON PROGRAMS



```
1
```

```
>
```

Panel A – Enter your source code here

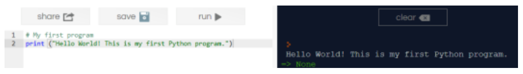
Panel B – Output of your code displays here

1

Let's Start Coding

- The most famous program in the world is a single line of code that prints "Hello World!" on the screen.
- After entering and running the "Hello World!" program, your program and output should look like Figure E-2.

FIGURE E-2: FIRST "HELLO WORLD!" PROGRAM



```
1  # My first program
2  print ("Hello World! This is my first Python program.")
```

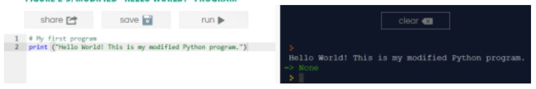
```
>
Hello World! This is my first Python program.
=> None
```

1

Let's Start Coding

- You can modify your program using the code editor. Figure E-3 shows the modified "Hello World!" program.

FIGURE E-3: MODIFIED "HELLO WORLD!" PROGRAM



```
1  # My first program
2  print ("Hello World! This is my modified Python program.")
```

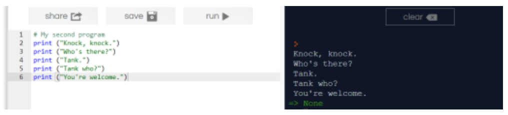
```
>
Hello World! This is my modified Python program.
=> None
>
```

1

Let's Start Coding

- Most programs contain more than one line of code. The example below in Figure E-4, demonstrates how to write a multiline"Knock Knock" joke in Python.

FIGURE E-4: CREATE A MULTI-LINE PROGRAM



```
1  # My second program
2  print ("Knock, knock.")
3  print ("Who's there?")
4  print ("Tank.")
5  print ("Tank who?")
6  print ("You're welcome.")
```

```
>
Knock, knock.
Who's there?
Tank.
Tank who?
You're welcome.
=> None
```

1

Let's Start Coding

- The Python programming language has its own **syntax**, which is a set of rules that defines how it can be written.
- A **comment** in Python is used by programmers to explain what the code does–it does not show up when the program runs.
- An important rule to remember is that a comment must begin with the # character and end on the physical line on which the # character has been typed.

1

Working with Strings

- When you write a sentence, you create a sequence of words that your reader will understand. Similarly, in programming you create a sequence of characters called a **string**, which can be made up of words, letters, punctuation marks, and numerals.
- For example, in your first program, line 2 contains this string:

```
"Hello World! This is my first Python program."
```

1

Working with Strings

- The term **concatenation** is used by programmers any time two or more characters are connected.
- Several strings can be connected using a symbol, such as the + symbol, as a **concatenation** operator. The example below uses the + symbol to concatenate two strings:

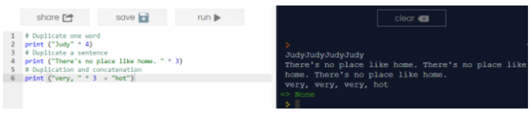
```
print("Hello World! " + "This is my first Python
program.")
```

1

Working with Strings

- Python gives programmers a shortcut for working with repeated strings. To print the same word more than once, just use the * symbol and the number of times you want it duplicated. Figure E-5 shows what your output should look like.

FIGURE E-5: DUPLICATION EXAMPLES



```
# Duplicate one word
print ("Judy" * 4)
# Duplicate a sentence
print ("There's no place like home. " * 3)
# Duplication and concatenation
print ("very, " * 3 + "hot")
```

```
>
JudyJudyJudyJudy
There's no place like home. There's no place like
home. There's no place like home.
very, very, very, hot
=> None
>
```

1

Using Python Keywords

- All programming languages have their own vocabulary, which is based on a set of keywords. Python has a small vocabulary of only 33 keywords, of which only about 10 are frequently used.

FIGURE E-6: PYTHON KEYWORDS

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	print
break	except	in	raise	

1

Debugging

- Programs must be tested to see if they work correctly. A programming error is called a **bug**.
- The process for tracking down bugs and correcting them is called **debugging**.
- Syntax errors and logic errors are the two most frequently encountered.

1

Debugging

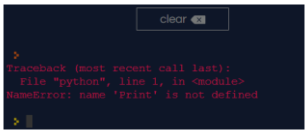
- A **syntax error** occurs when an instruction does not follow the rules of the programming language.
- **Some Python syntax rules are:**
  - Comments always start with a #.
  - Python is case sensitive.
  - Strings are delineated by quotation marks.
  - Keywords can only be used for their intended purpose.

1

Debugging

- If you receive an error message you can check the line of code where the error resides.
- Figure E-7 shows an error message generated in Python. Read the message carefully to identify your error.

FIGURE E-7: SYNTAX ERROR OUTPUT



```
>
Traceback (most recent call last):
  File "python", line 1, in <module>
NameError: name 'Print' is not defined

>
```

1

Section B: The Wacky Word Game

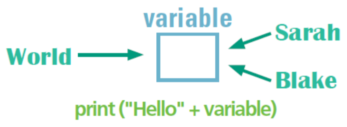
- Using Variables
- Objects and Classes
- Input
- Wacky Word Game
- Sharing Your Programs

1

Using Variables

- Technically, a **variable** is a named memory location that holds data specified by a programmer or entered by an end user.
- Programmers think of variables as empty boxes where data can be temporarily stored and used by a computer program.

FIGURE E-8: USING VARIABLES TO CUSTOMIZE YOUR PROGRAM



1

Using Variables

- A variable name should describe the information the variable is designed to store. For example, a good name for a variable that will contain a first name might be "firstname" or "first_name."

Here are some specific rules for naming variables in Python:

- The name must begin with a letter or an underscore character "_".
- The name can contain only letters, numbers, and the underscore character; no punctuation characters, special characters, or spaces are allowed.
- Python keywords, such as print and class, are not allowed as variable names.
- Every variable should have a unique name.

1

Using Variables

FIGURE E-9: INVALID VARIABLE NAMES IN PYTHON

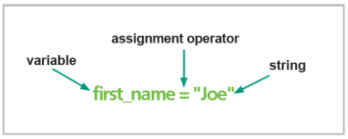
INVALID NAME	PROBLEM
`76trombones`	The name does not begin with a letter or an underscore.
`more$`	The name cannot contain a special character.
`last name`	The name cannot contain a space.
`class`	The name cannot contain a Python keyword.

1

Using Variables

- The process of creating a variable is sometimes referred to as *declaring a variable*.
- Putting data in a variable is referred to as *assigning a value* to it.

FIGURE E-10: ASSIGNING A VALUE TO A VARIABLE



1

Using Variables

- The type of data that a variable can hold is referred to as its **data type**.

FIGURE E-13: COMMON DATA TYPES

DATA TYPE	EXAMPLES
Integers	-2, -1, 0, 1, 2, 3, 4, 5
Floating-point numbers	-1.0, -0.5, 0.0, 1.0, 1.25
Strings	"Joe", "Hello!", "11 cats"

Objects and Classes

1

- Python is an **object-oriented** programming language, which is a language that enables the programmer to use objects to accomplish a program's goals.
- The **object-oriented paradigm** is based on objects and classes that can be defined and manipulated by program code.
- **Object-oriented programming** (OOP) is a style of programming that focuses on using objects to design and build applications.

Objects and Classes

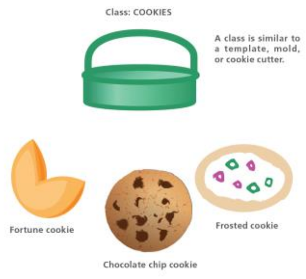
1

- An **object** is anything that can be seen, touched, or used; it can be a person, a place, or a thing.
- Every object in an OOP is crated from a **class**, which is a description or template that the computer uses to create the object.
- A **class attribute** defines the characteristics of a set of objects.
- An object created from a class is called an instance of a class and is said to be **instantiated** (created) from the class.

Objects and Classes

1

FIGURE E-15: CLASSES AND OBJECTS



Input

1

- Python uses the **input command** to get user input.
- The input command allows the program to display a message on the screen that asks a user to enter information; this message is called a **prompt**.
- For example, suppose you want a program to gather a user's name. You could write the following code:

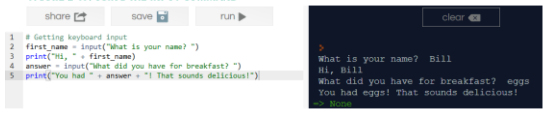
```
first_name = input("Please enter your name: ")
```

Input

1

- Writing prompts using the input command is a great way to relay instructions to your program's user. Figure E-17 demonstrates how to use the input command in your code.

FIGURE E-17: USING THE INPUT COMMAND



```
# Getting keyboard input
first_name = input("What is your name? ")
print("Hi, " + first_name)
answer = input("What did you have for breakfast? ")
print("You had " + answer + "! That sounds delicious!")
```

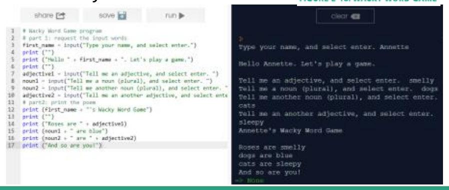
```
>
What is your name?  Bill
Hi, Bill
What did you have for breakfast?  eggs
You had eggs! That sounds delicious!
=> None
```

Wacky Word Game

1

- The Wacky Word Game uses variables, strings, and input. This program produces a game that prompts a player for a list of words and then asks the player to use them in a story or a poem. The output generates a nonsensical or comical story.

FIGURE E-18: WACKY WORD GAME



```
# Wacky Word Game program
# part 1: request the input words
first_name = input("Type your name, and select enter.")
print ("")
print ("Hello " + first_name + ". Let's play a game.")
print ("")
adjective1 = input("Tell me an adjective, and select enter. ")
noun1 = input("Tell me a noun (plural), and select enter. ")
noun2 = input("Tell me another noun (plural), and select enter. "
adjective2 = input("Tell me an another adjective, and select ente
# part2: print the poem
print (first_name + "'s Wacky Word Game")
print ("")
print ("Roses are " + adjective1)
print (noun1 + " are blue")
print (noun2 + " are " + adjective2)
print ("And so are you!")
```

```
>
Type your name, and select enter. Annette

Hello Annette. Let's play a game.

Tell me an adjective, and select enter.  smelly
Tell me a noun (plural), and select enter.  dogs
Tell me another noun (plural), and select enter.
cats
Tell me an another adjective, and select enter.
sleepy
Annette's Wacky Word Game

Roses are smelly
dogs are blue
cats are sleepy
And so are you!
=> None
```

1

Sharing Your Programs

- You may want to share your program code with your friends, instructors, etc.
- To do so, you can use repl.it’s **Share** option.
- You can also make a **screenshot** of the program and share it as a file.

1

Section C: Build Your Own Calculator

- Calculations
- Selection Structures
- Comparison and Logical Operators
- Comparing Strings

1

Calculations

- When a computer calculates an arithmetic operation it is called **computation**.
- To instruct the computer to perform a calculation, programmers use an **arithmetic expression**, which contains values (such as 2 and 3) and arithmetic operators (such as + and -).
- Figure E-19, on the next slide, illustrates a simple arithmetic expression and the symbols Python uses for arithmetic operators.

1

Calculations

FIGURE E-19: ARITHMETIC EXPRESSIONS

1 + 2 * 3
An arithmetic expression

OPERATOR	OPERATION	EXAMPLE	OUTPUT
+	Addition	4+3	7
-	Subtraction	7-5	2
*	Multiplication	4*5	20
/	Division	19/5	3
%	Modulus (Remainder)	19%5	4
**	Exponent	2**3	8

1

Calculations

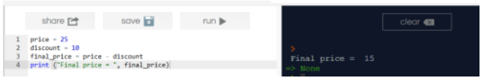
- The result of an arithmetic expression depends on the order in which Python performs the math.
- In mathematics, the **order of operations** is a collection of rules that dictate which procedures to perform first when calculating an arithmetic expression.
- In Python, the order of operations follows these rules:
  - Perform expressions in parentheses () first.
  - Resolve exponential notations, such as 2**2, which means 2^2.
  - Perform multiplication (*), division (/), and modulus (%) operations.
  - Perform addition (+) and subtraction (-) last.

1

Calculations

- Programmers frequently set up calculations by loading values into variables and then writing formulas using variables instead of numbers.
- This technique makes it easy to modify the numbers used in a calculation or get the numbers as input when a program runs.
- Figure E-20 shows a program with the variables “price” and “discount.”

FIGURE E-20: VARIABLES IN EXPRESSIONS



```
price = 25
discount = 10
final_price = price - discount
print ("Final price = ", final_price)
```

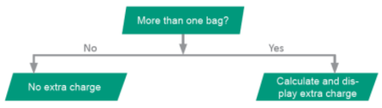
```
>
Final price =  15
=> None
```

1

Selection Structures

- A **selection control structure** tells a computer what to do based on whether a condition is true or false.
- You can think of a selection control as one or more paths in a program.
- Figure E-24 illustrates a simple branch using a checked bags example.

FIGURE E-24: A SIMPLE CONTROL STRUCTURE CREATES TWO PATHS



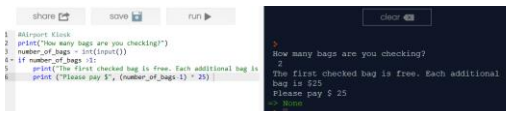
Using the *if* Command to Control Program Flow

1

Selection Structures

- An example of a selection control structure is the *if* command.
- Figure E-25 illustrates how an if statement works in a program using the airport kiosk checked bag example from Figure E-24.

FIGURE E-25: A SIMPLE IF STATEMENT



```
#Airport Kiosk
print("How many bags are you checking?")
number_of_bags = int(input())
if number_of_bags >1:
    print("The first checked bag is free. Each additional bag is
    print ("Please pay $", (number_of_bags-1) * 25)
```

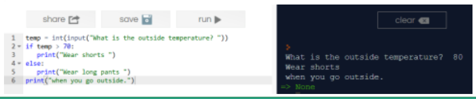
```
>
How many bags are you checking?
 2
The first checked bag is free. Each additional
bag is $25
Please pay $ 25
=> None
```

1

Selection Structures

- Frequently, programmers want one thing to happen when a condition is true and something else to happen when it is false.
- To accomplish this, they use the *if…else* statement.
- Figure E-26 illustrates how to program an if…else statement about weather conditions and clothing.

FIGURE E-26: IF...ELSE STATEMENT



```
temp = int(input("What is the outside temperature? "))
if temp > 70:
   print("Wear shorts ")
else:
   print("Wear long pants ")
print("when you go outside.")
```

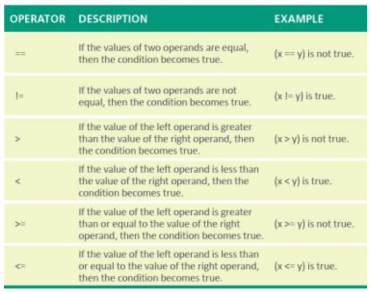
```
>
What is the outside temperature?  80
Wear shorts
when you go outside.
=> None
```

1

Comparison and Logical Operators

- A **comparison operator** is used in an expression to compare two values.
- The most commonly used comparison operators are >, < and ==.

FIGURE E-28: COMPARISON OPERATORS



OPERATOR	DESCRIPTION	EXAMPLE
==	If the values of two operands are equal, then the condition becomes true.	(x == y) is not true.
!=	If the values of two operands are not equal, then the condition becomes true.	(x != y) is true.
>	If the value of the left operand is greater than the value of the right operand, then the condition becomes true.	(x > y) is not true.
<	If the value of the left operand is less than the value of the right operand, then the condition becomes true.	(x < y) is true.
>=	If the value of the left operand is greater than or equal to the value of the right operand, then the condition becomes true.	(x >= y) is not true.
<=	If the value of the left operand is less than or equal to the value of the right operand, then the condition becomes true.	(x <= y) is true.

1

Comparison and Logical Operators

- The == operator is the **equality operator**; it is used for comparisons.
- The = symbol is the **assignment operator**; it is used to store values and strings in variables.
- **Two rules** come in handy when using comparison operators:
  - **First rule**: If an expression contains more than one comparison operator, the operators are evaluated from left to right in the expression.
  - **Second rule**: Comparison operators are evaluated after any arithmetic operators in an expression. For example, in 3 + 6 < 16 / 2, the two arithmetic operators will be evaluated first, and then the two resulting numbers will be compared.

1

Comparison and Logical Operators

- *If…else* statements can also contain logical operators.
- Python has three logical operators: AND, OR, and NOT.
- Python evaluates logical expressions as true or false, so they can be the basis for control structures that use *if* statements.

Comparing Strings

- When a program collects string input, such as a user's name, it can be used in expressions that become part of control structures.
- For example, a program might ask users if they know how to swim in order to enroll them in the appropriate swim class.
- Strings in Python are case sensitive, which means the string "Yes" is not the same as either the string "YES" or the string "yes".
- To avoid problems with case, you can use the upper() and lower() methods to convert string input to a known case.

Section D: Ask The Fortune Teller

- Repetition Control Structures
- Lists

Repetition Control Structures

- A repetition control structure allows programmers to write code that can repeatedly execute a statement or a series of statements.
- The section of code that repeats is referred to as a loop or an iteration.
- Python has two types of loops: the for-loop and the while-loop.
- For-loops make it easy to specify the number of repetitions in a loop.

Lists

- A list in Python is an ordered group of items that can be numbers or strings that are modifiable.
- The following are some examples of lists:

```
my_list = [1, 2, 3, 4, 5]
some_names = ["Bob", "Smith", "Jay"]
fruit_list = ["apple", "banana", "cantaloupe"]
```

Lists

- Lists are tools that programmers use to make certain programming tasks straightforward when combined with repetition.
- Lists can be used for mathematical operations, such as totaling the items in a list and placing the result in an accumulator.
- An **accumulator** is a numeric variable in which values are repetitively added.

Section E: Dogs and Cats

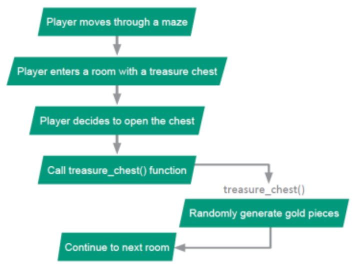
- Functions
- Methods

1 Functions

- In this Unit, you have used several of Python's built-in functions, such as print(), input(), and str().
- A programmer-defined function is typically a block of code that is part of a program but is not included in the main execution path.
- Figure E-42, on the next slide, illustrates how a programmer would visualize the structure of a program containing the treasure_chest() function.

1 Functions

FIGURE E-42: FUNCTIONS IN AN ADVENTURE PROGRAM


Player moves through a maze
Player enters a room with a treasure chest
Player decides to open the chest
Call treasure_chest() function
treasure_chest()
Randomly generate gold pieces
Continue to next room

1 Functions

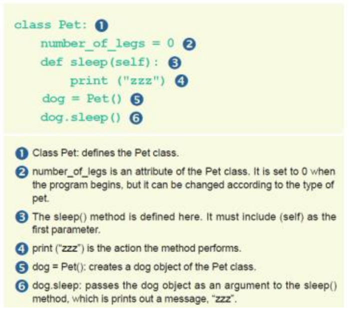
- **When using functions, keep the following in mind:**
  - Function blocks begin with the keyword *def* followed by the function name and parentheses ().
  - The parentheses can hold parameters. Make sure that the function call and the function definition have the same number of parameters.
  - The code block within every function starts with a colon (:) and is indented.
  - The function terminates with the last indented line of code.
  - The return statement passes data from the function to the main program.

1 Methods

- A **method** is a segment of code that defines an action belonging to a class.
- In Python, methods are essentially functions, but they are defined slightly differently; a method must always have an argument called *self* within the parentheses.
- When Python calls a method, it passes the current object to that method as the first parameter.
- Figure E-48, on the next slide, illustrates how this works.

1 Methods

FIGURE E-48: A METHOD FOR THE PET CLASS



```
class Pet: ❶
    number_of_legs = 0 ❷
    def sleep(self): ❸
        print ("zzz") ❹
    dog = Pet() ❺
    dog.sleep() ❻
```

❶ Class Pet: defines the Pet class.
❷ number_of_legs is an attribute of the Pet class. It is set to 0 when the program begins, but it can be changed according to the type of pet.
❸ The sleep() method is defined here. It must include (self) as the first parameter.
❹ print ("zzz") is the action the method performs.
❺ dog = Pet(): creates a dog object of the Pet class.
❻ dog.sleep: passes the dog object as an argument to the sleep() method, which is prints out a message, "zzz".

1 Methods

- In object-oriented jargon, **inheritance** refers to passing certain characteristics from one class to other classes.
- A **superclass** is the class from where attributes and methods can be in inherited.
- A **subclass** inherits attributes and methods from a superclass.
- **Polymorphism**, sometimes called overloading, is the ability to redefine a method in a subclass. It enables programmers to create a single, more generic name for a method that behaves in unique ways for different classes.

NEW PERSPECTIVES

Unit E Complete

Computer Concepts 2016

ENHANCED EDITION