PYTHON BOOT CAMP

Module 2:
Elementary Programming

Motivations

- In the preceding chapter, you learned how to create, compile, and run a Python program.
- Starting from this chapter, you will learn how to solve practical problems programmatically.
  - Through these problems, you will learn Python primitive data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.

Objectives

- To write programs that perform simple computations (§2.2).
- To obtain input from a program's user by using the **input** function (§2.3).
- To use identifiers to name variables (§2.4).
- To assign data to variables (§2.5).
- To define named constants (§2.6).
- To use the operators **+**, **-**, *****, **/**, **//**, **%**, and ****** (§2.7).
- To write and evaluate numeric expressions (§2.8).
- To use augmented assignment operators to simplify coding (§2.9).
- To perform numeric type conversion and rounding with the **int** and **round** functions (§2.10).
- To obtain the current system time by using **time.time()** (§2.11).
- To describe the software development process and apply it to develop the loan payment program (§2.12).
- To compute and display the distance between two points (§2.13).

Writing a Simple Program

- Write a program that will calculate the area of a circle.
- Think of every problem as having two main parts, or phases*:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

*On these smaller problems, the problem-solving phase may seem silly (unnecessary), but it is fully necessary on more complex problems.

Writing a Simple Program

- Write a program that will calculate the area of a circle.
- **Step 1**: Design your algorithm
  1. Get the radius of the circle.
  2. Compute the area using the following formula:
     - area = radius x radius x π
  3. Display the result

Writing a Simple Program

- Write a program that will calculate the area of a circle.
- **Step 2**: Implementation (code the algorithm)

```
# Step 1: get radius

# Step 2: calculate area

# Step 3: display the result
```

  - Notice that we start by writing **comments** for each part

Writing a Simple Program

- Write a program that will calculate the area of a circle.
- **Step 2**: Implementation (code the algorithm)
  - In order to store the radius, the program must declare a symbol called a **variable**.
  - A variable represents a value stored in the computer's memory
  - You should choose good names for variables
    - Do not choose "x" or "y"…these have no meaning
    - Choose names with meaning…"area" or "radius"

Writing a Simple Program

- Write a program that will calculate the area of a circle.
- **Step 2**: Implementation (code the algorithm)
  - What value do you want to store in *radius*?
  - What about *area*?
    - Integer? Real number? Something else maybe?
  - The variable's **data type** is the kind of data that you can store in that particular variable.
  - Python automatically figures out the data type according to the value assigned to the variable

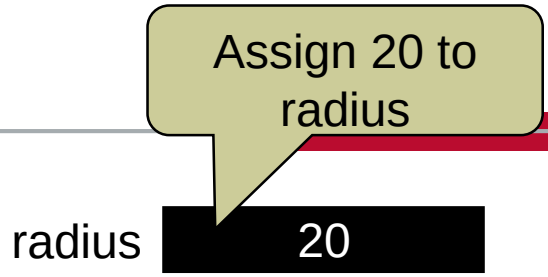
Writing a Simple Program

- Write a program that will calculate the area of a circle.
- **Step 2**: Implementation (code the algorithm)

```
# Step 1: get radius
radius = 20 # radius is now 20

# Step 2: calculate area
area = radius * radius * 3.14159

# Step 3: display the result
print("The area for the circle of radius", radius, "is", area)
```

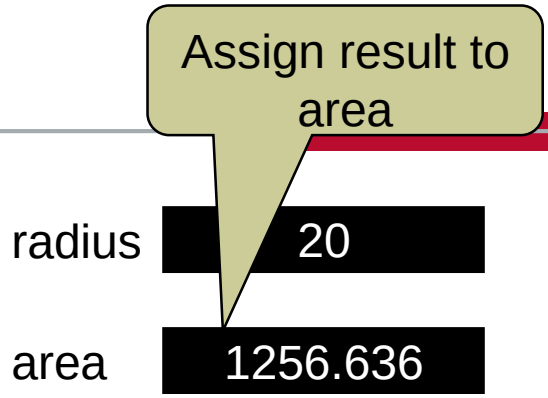
Trace a Program Execution

```
# Assign a radius
radius = 20 # radius is now 20
# Compute area
area = radius * radius * 3.14159
# Display results
print("The area for the circle of radius " +
    str(radius) + " is " + str(area))
```



Trace a Program Execution

```
# Assign a radius
radius = 20 # radius is now 20
# Compute area
area = radius * radius * 3.14159
# Display results
print("The area for the circle of radius“,
    radius, " is "area)
```

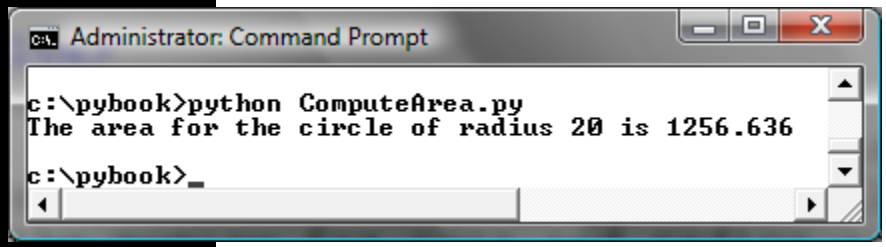


Trace a Program Execution

```
# Assign a radius
radius = 20 # radius is now 20
# Compute area
area = radius * radius * 3.14159
# Display results
print("The area for the circle of radius",
    radius, "is", area)
```

print a message to the console

radius	20
area	1256.636



Writing a Simple Program

- Discussion:
  - Variables such as **`radius`** and **`area`** refer to memory locations
  - Each variable has a name that refers to a value
  - And you can assign a value as shown below

    ```
    radius = 20
    ```

    - Here, we assign the value, 20, into the variable, `radius`

Writing a Simple Program

- Discussion:
  - The following table shows the value in memory for the variables area and radius as the program is executed.

line#	radius	area
2	20	
5		1256.636

  - This method of reviewing a program is called "*tracing a program*".
  - Helps you to understand how programs work.

Check Point

- Translate the following algorithm into Python code:
  - Step 1: Use a variable named miles with initial value 100.
  - Step 2: Multiply miles by 1.609 and assign it to a variable named kilometers.
  - Step 3: Display the value of kilometers.
  - What is kilometers after Step 3?

Reading Input from the Console

- In the last example, the radius was fixed.
- Your program can be better, and more interactive, by letting the user enter the radius.
- Python uses the **input** function for this purpose

```
variable = input("Enter a value: ")
```

  - ***By default, Python understands any input to be a string value***
  - Example: the user enters the number 81
    - Even though we know/view this as a number
    - Python will see it as a string of two characters
      - An '8' followed by a '1'…Python does not see it as the number 81, by default

Reading Input from the Console

- Function **eval**:
  - You can use the **eval** function to have Python evaluate (or in this case, convert) what is inside the string
  - Examples:
    - `eval("34.5")` returns 34.5
    - `eval("345")` returns 345
    - `eval("3 + 4")` returns 7
    - `eval("51 + (54 * (3 + 2))")` returns 321
  - Now we can revisit our previous solution and ask the user to enter a radius…

Program 2: Compute Area with Console Input

- Write a program that will calculate the area of a circle…but this time using user input for radius.
- Remember
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

    - We've already done the problem-solving phase
    - Copy your last code and paste it into a new Thonny program
    - Edit accordingly

Program 2: Compute Area with Console Input

LISTING 2.2 ComputeAreaWithConsoleInput.py

```
1  # Prompt the user to enter a radius
2  radius = eval(input("Enter a value for radius: "))
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius", radius, "is", area)
```

```
Enter a value for radius: 2.5 ↵Enter
The area for the circle of radius 2.5 is 19.6349375
```

```
Enter a value for radius: 23 ↵Enter
The area for the circle of radius 23 is 1661.90111
```

Program 3: Compute Average

- Write a program to get three values from the user and compute their average.
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Program 3: Compute Average

- Write a program to get three values from the user and compute their average.
- **Step 1**: Design your algorithm
  1. Get three numbers from the user.
     - Use `input` and `eval` functions
     - You can ask for each number on a different line
     - "Please enter the first number: "
     - "Please enter the second number: ", and so on
  2. Compute the average of the three numbers:
     - average = (num1 + num2 + num3) / 3
  3. Display the result

Program 3: Compute Average

- Write a program to get three values from the user and compute their average.
- **Step 2**: Implementation (code the algorithm)

```
# Step 1: ask user to enter three values

# Step 2: calculate average

# Step 3: display the results
```

Program 3: Compute Average

LISTING 2.3 ComputeAverage.py

```
 1  # Prompt the user to enter three numbers
 2  number1 = eval(input("Enter the first number: "))
 3  number2 = eval(input("Enter the second number: "))
 4  number3 = eval(input("Enter the third number: "))
 5
 6  # Compute average
 7  average = (number1 + number2 + number3) / 3
 8
 9  # Display result
10  print("The average of", number1, number2, number3,
11      "is", average)
```

```
Enter the first number: 1 Enter
Enter the second number: 2 Enter
Enter the third number: 3 Enter
The average of 1  2  3 is 2.0
```

Program 3: Compute Average

- Discussion:
  - If the user enters something other than a number, your program would give an error
    - We'll discuss how do to deal with this later
  - Statements are usually only one line
    - But in the last example, the print statement was over two lines
    - Why was this okay?
    - Because Python scans the print statement and knows it is not complete until it finds the closing parenthesis
    - So we say these two lines are joined implicitly

Program 3: Compute Average

- Discussion:
  - Using the line continuation symbol (\)
    - In same cases, Python cannot determine the end of a statement if it is written across multiple lines
    - Example:

```
sum = 1 + 2 + 3 + 4 +
      5 + 6
```

    - That would cause an error
    - If some reason, you absolutely needed to write a single statement across multiple lines, you can use the backslash as follows

```
sum = 1 + 2 + 3 + 4 + \
      5 + 6
```

Identifiers

- What is an identifier?
  - ***Identifiers*** are the *names* that identify elements of your program, *such as variables and functions*
  - More specifically:
    - An identifier is a sequence of characters that consists of letters, digits, underscores (_), and asterisk (*).
    - An identifier must start with a letter or an underscore. It cannot start with a digit.
    - An identifier cannot be a reserved word. (See Appendix A, "Python Keywords," for a list of reserved words.) Reserved words have special meanings in Python, which we will later.
    - An identifier can be of any length.

Identifiers

- Examples of legal identifiers:
  - area, radius, number1
- Example of illegal identifiers:
  - 2A, d+4
- Python is case sensitive
  - area, Area, and AREA are all different identifiers

Variables

- Variables are used to represent values that may be changed in the program.
  - In the previous programs, we used variables to store values
    - area, radius, average, etc.
- They are called variables because their values can be changed!

Variables

```
1  # Compute the first area
2  radius = 1.0                                    radius → 1.0
3  area = radius * radius * 3.14159                  area → 3.14159
4  print("The area is", area, "for radius", radius)
5
6  # Compute the second area
7  radius = 2.0                                    radius → 2.0
8  area = radius * radius * 3.14159                  area → 12.56636
9  print("The area is", area, "for radius", radius)
```

- Discussion:
  - radius is initially 1.0 (line 2)
  - then changed to 2.0 (line 7)
  - area is computer as 3.14159 (line 3)
  - then changed to 12.56636 (line 8)

Assignment Statements

- When we create a variable in Python, we initialize it with some starting value
- And we do this with the = sign
  - which is also known as the **assignment operator**
- Assignment Operator (=):
  - The syntax is as follows:

```
variable = expression
```

    - An expression represents a computation involving values, variables, and operators that, taken together, evaluate to a value
    - The assignment operator takes whatever is on the right side and **saves it** (or, "**assigns it**") to the variable on the left

Assignment Statements

- Examples:

```
y = 1                       # Assign 1 to variable y
radius = 1.0                # Assign 1.0 to variable radius
x = 5 * (3 / 2) + 3 * 2     # Assign the value of the expression to x
x = y + 1                   # Assign the addition of y and 1 to x
area = radius * radius * 3.14159 # Compute area
```

  - Variables can be used in an expression
  - And the same variable can be used on both sides of the expression
    - Example:

      `x = x + 1`

      - Here, the result of '`x + 1`' is assigned into (saved into) the variable `x`
      - So if x has the value of 1 before this statement, what would it be after?
      - x would equal 2 after the statement

- Continue here

Assignment Statements

- Note:
  - Consider the following:

```
x = 2 * x + 1
```

    - In mathematics, this would be an equation
    - **In Python, this is an assignment statement**
    - The "`2 * x + 1`" is evaluated and then saved into `x`
    - How? It's evaluated using whatever value x had previously
  - You can also assign a value to multiple variables at once:

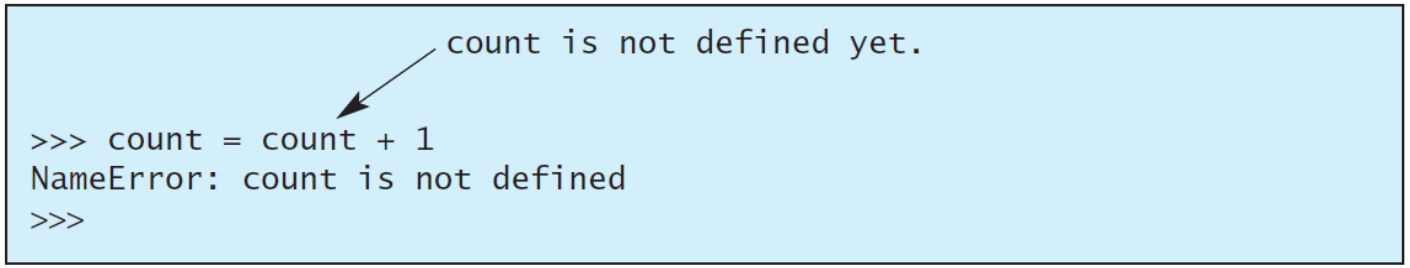
```
i = j = k = 1
```

    - This is the same as:

```
i = 1
j = 1
k = 1
```

Assignment Statements

- Note:
  - Variables must be created before they are used!



    - This can be fixed as follows:

```
>>> count = 1  # count is not created
>>> count = count + 1  # Now increment count
>>>
```

Swapping Two Values

- Consider that you have two variables:
  - `x` and `y`
- These two variables both have integer values inside them
- You'd like to write a few lines of code to SWAP what is inside those variables
  - So `x` should end up having the value that was inside `y`
  - And `y` should end up having the value that was inside `x`
- Try to do this yourselves…

Simultaneous Assignment

- A cool feature of Python!
- But first, what is simultaneous assignment?
  - Syntax:

    ```
    var1, var2, ..., varn = exp1, exp2, ..., expn
    ```

    - This tells Python to evaluate all the expressions on the right
    - Then tells Python to save them into the corresponding variables on the left

  - Really, not that useful at first glance
  - But consider a VERY famous example…
    - (the one you all just did yourselves…SWAP)

Simultaneous Assignment

- Famous example: Swapping Values!
  - a very common operation in programming
  - Consider two variables, `x` and `y`
    - Assume `x = 1` and `y = 2`
  - How can we swap their values?
  - Answer: we need a temporary variable
    - We'll call it `temp` cause we're super-duper original
  - Here's the code:

```
>>> x = 1
>>> y = 2
>>> temp = x  # Save x in a temp variable
>>> x = y     # Assign the value in y to x
>>> y = temp  # Assign the value in temp to y
```

Simultaneous Assignment

- Famous example: Swapping Values!
  - But check out how easy this is to do using simultaneous assignment in Python:

```
>>> x, y = y, x # Swap x with y
```

  - That's it!
    - I know, I know…since most of you haven't coded before…this may not seem like a big deal
    - But trust me, it is!

- Now let's redo the average example reading in all three values in one go…

Program 3: Compute Average with Simultaneous Assignment

LISTING 2.4 ComputeAverageWithSimultaneousAssignment.py

```
1  # Prompt the user to enter three numbers
2  number1, number2, number3 = eval(input(
3      "Enter three numbers separated by commas: "))
4
5  # Compute average
6  average = (number1 + number2 + number3) / 3
7
8  # Display result
9  print("The average of", number1, number2, number3
10     "is", average)
```

```
Enter three numbers separated by commas: 1, 2, 3 ↵Enter
The average of 1  2  3 is 2.0
```

Named **Constants**

- A named constant is an identifier that represents a **permanent value**.
  - The value of a variable can change during execution of a program.
  - However, a *named constant*, or simply ***constant***, represents a permanent data that **never** changes.
  - PI in the example above was, in fact, a constant
    - PI never changes
  - In Python, there is no special syntax for constants
    - You just use regular variables
    - But we denote it as a constant by using ALL UPPERCASE

Named Constants

- Revisiting Problem 1:

```
# Assign a radius
radius = 20 # radius is now 20

# Compute area
PI = 3.14159
area = radius * radius * PI

# Display results
print("The area for the circle of radius", radius, "is", area)
```

Named Constants

- Benefits of named constants:
  1. You don't have to repeatedly type the same value if it is used multiple times.
  2. If you have to change the constant's value (e.g., from 3.14 to 3.14159 for PI), you need to change it only in a single location in the source code.
  3. Descriptive names make the program easy to read.
     - In this case, UPPERCASE signals the reader that the "variable" in fact should be understood as a constant

Numeric Data Types and Operators

- What is a data type?
  - Information stored in a computer generally called data
  - There are two types of numeric data:
    1. Integers
    2. Real numbers
  - Integers (`int` for short) are represented using whole numbers
  - Real numbers are represented with a fractional part
    - Insider the computer real numbers are represented as *floating-point values* (aka ***floats***)

Numeric Data Types and Operators

- Distinguishing between `int`s and `float`s?
  - If a number has a decimal point, it is a float. Simple.
  - Even if there is only a zero after the decimal, it is still a float.
    - So 1.0 is considered a float, but 1 is an integer
  - And this is important because these two numbers are stored different internally

- The following slide shows the Python operators for numeric data types

Numeric Data Types and Operators

TABLE 2.1 Numeric Operators

Name	*Meaning*	*Example*	*Result*
`+`	Addition	`34 + 1`	`35`
`-`	Subtraction	`34.0 - 0.1`	`33.9`
`*`	Multiplication	`300 * 30`	`9000`
`/`	Float Division	`1 / 2`	`0.5`
`//`	Integer Division	`1 // 2`	`0`
`**`	Exponentiation	`4 ** 0.5`	`2.0`
`%`	Remainder	`20 % 3`	`2`

Numeric Data Types and Operators

- The /, //, and ** operators:
  - The / performs "standard" division, aka floating-point division:

```
>>> 4 / 2
2.0
>>> 2 / 4
0.5
>>>
```

  - The // performs integer division:

```
>>> 5 // 2
2
>>> 2 // 4
0
>>>
```

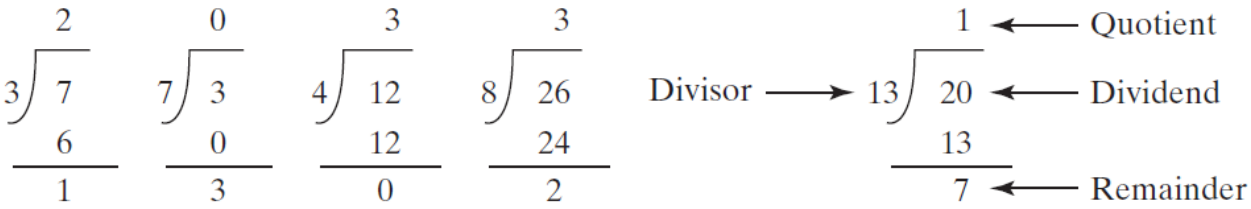
Numeric Data Types and Operators

- Exponents in Python
  - Given two numbers, `a` and `b`, Python can compute a^b
    - This is `a` raised to the `b` power
  - How?
    - You use the exponent operator: `**`
      - This is two asterics next to each other
    - So, `a ** b`
  - Examples:

```
>>> 2.3 ** 3.5
18.45216910555504
>>> (-2.5) ** 2
6.25
>>>
```

Numeric Data Types and Operators

- The % Operator
  - This is famously known as the **mod** operator
  - It calculates the remainder after classical/integer division
  - Examples:
    - 7 % 3 = 1
    - 3 % 7 = 3
    - 12 % 4 = 0
    - 26 % 8 = 2
    - 20 % 13 = 7
    - 19 % 5 = 4



Numeric Data Types and Operators

- The % Operator in practice
  - mod is VERY useful in programming!
  - Example:
    - We can use it to test if an integer is even or odd
    - Let's pause and try to figure out, in groups, how we can use mod to determine if a number is even or odd…
    - An even number % 2 is always 0
      - Right?
      - Cuz an even number divided by 2 always has a remainder of 0!
    - And an odd number % 2 is always 1
      - Cuz an odd number divided by 2 always has a remainder of 1!

Numeric Data Types and Operators

- The % Operator in practice
  - mod is VERY useful in programming!
  - Another example:
    - Given 195 seconds, how many minutes and how many seconds
      - Some quick math done in the head…3 minutes is 180 seconds
      - So we have a full 3 minutes and then 15 seconds left
    - We can calculate this using two operators:
      - Integer division: //
      - And mod: %
    - 195 // 60 = 3
      - This gives us the quantity of full minutes found within 195 seconds
    - 195 % 3 = 15
      - This gives us the remainder after dividing 195 by 3

Program 4: Display Time

- Write a program to prompt the user for a number of seconds, and then display how many minutes and seconds are obtained.
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Program 4: Display Time

- Write a program to prompt the user for a number of seconds, and then display how many minutes and seconds are obtained.
- **Step 1**: Design your algorithm
  1. Get total seconds from the user.
     - Use `input` and `eval` functions
  2. Compute the minutes and the remaining seconds
     - Using // and % as shown on previous pages
  3. Display the result

Program 4: Display Time

- Write a program to prompt the user for a number of seconds, and then display how many minutes and seconds are obtained.
- **Step 2**: Implementation (code the algorithm)

LISTING 2.5 DisplayTime.py

```
1  # Prompt the user for input
2  seconds = eval(input("Enter an integer for seconds: "))
3
4  # Get minutes and remaining seconds
5  minutes = seconds // 60      # Find minutes in seconds
6  remainingSeconds = seconds % 60   # Seconds remaining
7  print(seconds, "seconds is", minutes,
8      "minutes and", remainingSeconds, "seconds")
```

```
Enter an integer for seconds: 500 ↵ Enter
500 seconds is 8 minutes and 20 seconds
```

Numeric Data Types and Operators

Check Point

- What are the results of the following expressions?

Expression	Result
`42 / 5`	`8.4`
`42 // 5`	`8`
`42 % 5`	`2`
`40 % 5`	`0`
`1 % 2`	`1`
`2 % 1`	`0`
`45 + 4 * 4 - 2`	`59`
`45 + 43 % 5 * (23 * 3 % 2)`	`48`
`5 ** 2`	`25`
`5.1 ** 2`	`26.01`

Evaluating Expressions and Operator Precedence

- How to evaluate Python expressions?
  - Simple: the same way as arithmetic expressions!
  - Given the following expression:

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

  - We can directly translate it to Python as follows:

```
(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x +
9 * (4 / x + (9 + x) / y)
```

  - In summary:
    - You can safely apply classical arithmetic rules when evaluating a Python expression!

Evaluating Expressions and Operator Precedence

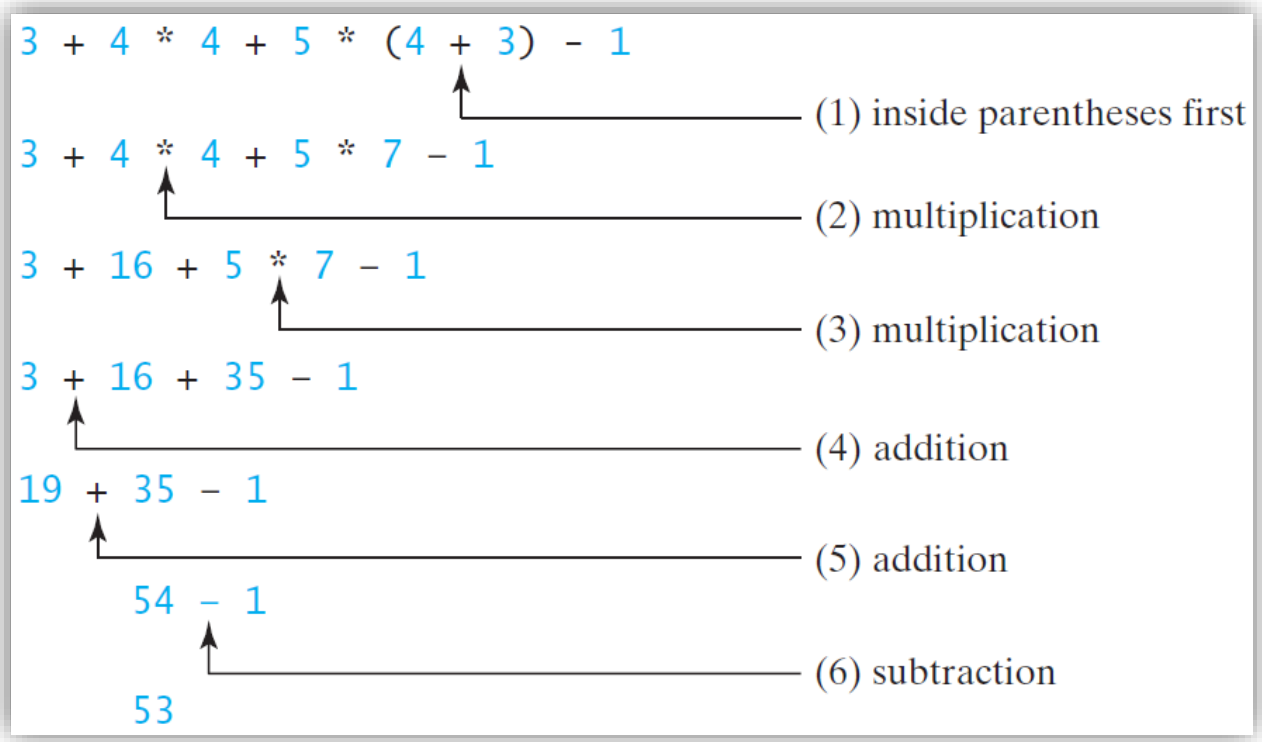
- Evaluating expressions is straightforward
  - But reminders are still helpful!
    - Operators contained within pairs of parentheses are evaluated first.
    - Parentheses can be nested, in which case the expression in the inner parentheses is evaluated first.
    - When more than one operator is used in an expression, the following operator precedence rule is used to determine the order of evaluation.

Evaluating Expressions and Operator Precedence

- Evaluating expressions is straightforward
  - But reminders are still helpful!
    - Exponentiation (**) is applied first.
    - Multiplication (*), float division (/), integer division (//) , and remainder operators (%) are applied next.
      - If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.
    - Addition (+) and subtraction (-) operators are applied last.
      - If an expression contains several addition and subtraction operators, they are applied from left to right.

Evaluating Expressions and Operator Precedence

- Evaluating expressions is straightforward
  - An example:



Evaluating Expressions and Operator Precedence

Check Point

- Write the following as a Python expression:

$$\frac{4}{3(r+34)} - 9(a+bc) + \frac{3+d(2+a)}{a+bd}$$

- Solution:

```
4.0 / (3.0 * (r + 34)) - 9 * (a + b * c) +
(3.0 + d * (2 + a)) / (a + b * d)
```

Augmented Assignment Operators

- Idea:
  - We like shortcuts!
    - Most programming languages offer augmented assignment operators
  - What are they?
    - Very often the current value of a variable is used, modified, and then reassigned back to the same variable
    - Example:

```
count = count + 1
```

    - Python allows you to combine the addition and assignment into one operator by using an augmented assignment operator

```
count += 1
```

      - specifically, this one is called an addition assignment operator

Augmented Assignment Operators

- A listing of augmented assignment operators:

TABLE 2.2 Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Float division assignment	`i /= 8`	`i = i / 8`
`//=`	Integer division assignment	`i //= 8`	`i = i // 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`
`**=`	Exponent assignment	`i **= 8`	`i = i ** 8`

  - Note: there are no spaces between op. and assignment
    - + = or * = would be wrong…it should be += or *=

Augmented Assignment Operators

Check Point

- Assume a = 1 and that each of the following expressions is independent. What are the results of the following?

```
a += 4          = 5
a -= 4          = -3
a *= 4          = 4
a /= 4          = 0.25
a //= 4         = 0
a %= 4          = 1
a = 56 * a + 6  = 62
```

Type Conversions and Rounding

- What is meant by type conversions?
  - We discussed data types
    - int and float
  - Well, we can convert one to another
    - In fact, if, for example, we add an `int` and a `float`, Python will automatically convert the `int` to a `float`
      - So `3 * 4.5` is converted to `3.0 * 4.5`
  - And sometimes it is helpful to get just the integer part of a real number…you can use the `int` function for this:

```
>>> value = 5.6
>>> int(value)
5
>>>
```

Note: the fractional part is truncated, not rounded up or down.

Type Conversions and Rounding

- And now rounding…
  - We all know what rounding is
  - Python gives us the **`round`** function to round numbers:

```
>>> value = 5.6
>>> round(value)
6
>>>
```

  - Note:
    - The **`int`** and **`round`** functions do not change the variable itself:

```
>>> value = 5.6
>>> round(value)
6
>>> value
5.6
>>>
```

Type Conversions and Rounding

- A note on `int` vs `eval`:
  - The `int` function can be used to convert a string to an integer:
    - `int("34")` returns the integer value `34`
  - This means we can use `int` or `eval`
  - Which is better?
    - That's debatable…but usually those in the "`int`" camp win
    - Why?
    - The `int` function is specific:
      - You, the programmer, are instructing Python exactly what to give
      - The `eval` function leaves it up to the discretion of Python
    - At the same time, `eval` can also be used to evaluate expressions, something that `int` cannot do

Program 5: Sales Tax

- Write a program to prompt the user for a sales amount and then display the sales tax due with exactly two digits after the decimal place.
  - Assume tax is 6%
    - Or you can ask the user to enter the sales tax percentage
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Program 5: Sales Tax

- Write a program to prompt the user for a sales amount and then display the sales tax due with exactly two digits after the decimal place.
- **Step 1**: Design your algorithm
  1. Get sales amount from the user.
     - Use `input` and `eval` functions
  2. Compute the sales tax with exactly two decimal places
     - The calculation is easy
     - But how to get two decimal places…
  3. Display the result

Program 5: Sales Tax

- Write a program to prompt the user for a sales amount and then display the sales tax due with exactly two digits after the decimal place.
- **Step 1**: Design your algorithm
  - How to get exactly two decimal places?
    - Consider the number `12.8742`
    - How can we isolate just the `12.87`?
      - Using only what we know till now…
    - Solution:
      - Multiply the number by 100
      - So we get `1287.42`
      - Now use the int function: `int(1287.42)` returns `1287`
      - Finally, divide by 100: `1287/100 = 12.87`

Program 5: Sales Tax

- Write a program to prompt the user for a sales amount and then display the sales tax due with exactly two digits after the decimal place.
- **Step 2**: Implementation (code the algorithm)

LISTING 2.6 SalesTax.py

```
1  # Prompt the user for input
2  purchaseAmount = eval(input("Enter purchase amount: "))
3
4  # Compute sales tax
5  tax = purchaseAmount * 0.06
6
7  # Display tax amount with two digits after decimal point
8  print("Sales tax is", int(tax * 100) / 100.0)
```

```
Enter purchase amount: 197.55 ↵Enter
Sales tax is 11.85
```

Type Conversions and Rounding

Check Point

- Are the following statements valid? If so, show the printed value.

```
value = 4.6
print(int(value))              4
print(round(value))            5
print(eval("4 * 5 + 2"))       22
print(int("04"))               4
print(int("4.5"))              error
print(eval("04"))              error
```

Program 6: Coin Change Problem

- Write a program that prompts the user for a "change" amount and displays the correct coins to be given in change.
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Program 6: Coin Change Problem

- Write a program that prompts the user for a "change" amount and displays the correct coins to be given in change.
- **Step 1**: Problem-solving Phase
  1. Prompt user for the change amount.
     - Use `input` and `int` functions
  2. Compute the correct type and number of coins
     - But how?
  3. Display the result

Program 6: Coin Change Problem

- Write a program that prompts the user for a "change" amount and displays the correct coins to be given in change.
- **Step 1**: Problem-solving Phase
  - Assume the user enters 93 cents
  - What are the correct coins returned:
    - 3 quarters
    - 1 dime
    - 1 nickel
    - 3 pennies
  - Cool…so how do we make that happen

Program 6: Coin Change Problem

- Write a program that prompts the user for a "change" amount and displays the correct coins to be given in change.
- **Step 1**: Problem-solving Phase
  - Calculating coins returned for 93 cents:
    - 3 quarters
      - How we get this: 93 // 25 = 3
      - Remaining pennies: 93 % 25 = 18
    - 1 dime
      - How we get this: 18 // 10 = 1
      - Remaining pennies: 18 % 10 = 8

Program 6: Coin Change Problem

- Write a program that prompts the user for a "change" amount and displays the correct coins to be given in change.
- **Step 1**: Problem-solving Phase
  - Calculating coins returned for 93 cents:
    - 1 nickel
      - How we get this: 8 // 5 = 1
      - Remaining pennies: 8 % 5 = 3
    - 3 pennies
      - How we get this: this is just the remaining pennies from answer above
  - So now time to translate into code…

Program 6: Coin Change Problem

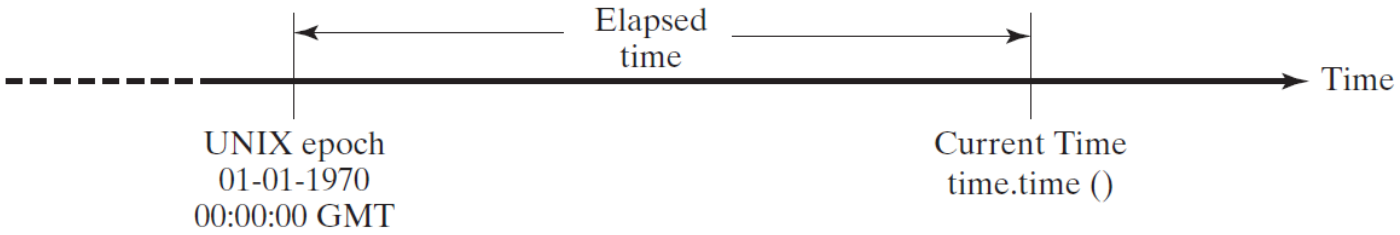
```
# Step 1: get change amount from user
change = int(input("Please enter the change to be given: "))

# Step 2: determine type and quantity of coins

# Calculate quarters
quarters = change // 25
pennies_remaining = change % 25

# Calculate dimes
dimes = pennies_remaining // 10
pennies_remaining = pennies_remaining % 10

# Calculate nickels
nickels = pennies_remaining // 5
pennies = pennies_remaining % 5

# Step 3: display the results
print(change, "cents should be given in change as:")
print("\t", quarters, "quarters")
print("\t", dimes, "dimes")
print("\t", nickels, "nickels")
print("\t", pennies, "pennies")
```

Program 7: Display Current Time

- Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Program 7: Display Current Time

- Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
- **Step 1**: Problem-solving Phase
  - Python gives us a `time()` function
    - This returns us the number of milliseconds elapsed since the time 00:00:00 on January 1, 1970 GMT
      - This is known as the UNIX epoch
        - The epoch is the point where time starts
        - 1970 was the year UNIX started



Program 7: Display Current Time

- Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
- **Step 1**: Problem-solving Phase
  - Python gives us a `time()` function
  - Example:
    - time.time() may return `1285543663.205`
    - This means `1285543663` seconds and `205` milliseconds
  - So…how exactly is this helpful?
    - Well, if we know the exact number of milliseconds since January 1, 1970, can we do something with that?
      - Sure…we can calculate the number of minutes, hours, days, months, years…

Program 7: Display Current Time

- Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
- **Step 1**: Problem-solving Phase
  - Algorithm:
    1. Obtain current # of milliseconds by invoking `time.time()`:
       - Example: 1203183068.328
    2. Obtain total seconds, `total_seconds`, using the `int` function:
       - Example: int(1203183068.328) = 1203183068
    3. Compute the current second by using mod
       - total_seconds % 60
       - 1203183068 % 60 = 8, which is the current second

Program 7: Display Current Time

- Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
- **Step 1**: Problem-solving Phase
  - Algorithm:
    4. Obtain total minutes, `total_minutes`, by dividing `total_seconds` by 60
       - 1203183068 // 60 = 20053051 minutes
    5. Compute the current minute by using mod
       - total_minutes % 60
       - 20053051 % 60 = 31, which is the current minute
    6. Obtain total hours, `total_hours`, by dividing `total_minutes` by 60
       - 20053051 // 60 = 334217 hours

Program 7: Display Current Time

- Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
- **Step 1**: Problem-solving Phase
  - Algorithm:
    7. Compute the current hour from `total_hours` % 24
       - 334217 % 24 = 17, which is the current hour
  - So we have many steps here
    - But once the algorithm is written about (above) and fully understood…translation to code is straightforward…

Program 7: Display Current Time

```
import time

current_time = time.time() # Get current time

# Obtain the total seconds since midnight, Jan 1, 1970
total_seconds = int(current_time)

# Get the current second
current_second = total_seconds % 60

# Obtain the total minutes
total_minutes = total_seconds // 60

# Compute the current minute in the hour
current_minute = total_minutes % 60

# Obtain the total hours
total_hours = total_minutes // 60

# Compute the current hour
current_hour = total_hours % 24

# Display results
print("Current time is", current_hour, ":", current_minute, ":", current_second, "GMT")
```

Program 8: Compute Distance

- Write a program that prompts the user to enter two points and calculates the distance between them.
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Program 8: Compute Distance

- Write a program that prompts the user to enter two points and calculates the distance between them.
- **Step 1**: Problem-solving Phase
  - This problem is rather straightforward
  - Main thing is to remember the 'ole distance formula:
    - $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
  - And how do we calculate the square root?
    - We use the exponent operator: **
    - Example: square root of 25: `25 ** 0.5 = 5`

Program 8: Compute Distance

- Write a program that prompts the user to enter two points and calculates the distance between them.
- **Step 2**: Implementation

LISTING 2.9 ComputeDistance.py

```
1  # Enter the first point with two float values
2  x1, y1 = eval(input("Enter x1 and y1 for Point 1: "))
3
4  # Enter the second point with two float values
5  x2, y2 = eval(input("Enter x2 and y2 for Point 2: "))
6
7  # Compute the distance
8  distance = ((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)) ** 0.5
9
10 print("The distance between the two points is", distance)
```

```
Enter x1 and y1 for Point 1: 1.5, -3.4 ↵Enter
Enter x2 and y2 for Point 2: 4, 5 ↵Enter
The distance between the two points is 8.764131445842194
```

Program 8: Compute Distance

- Write a program that prompts the user to enter two points and calculates the distance between them.
- Discussion:
  - We can also use the turtle to:
    - Display our points
    - Draw a line between them
    - Display the distance
  - Check it out…

Program 8: Compute Distance

LISTING 2.10 ComputeDistanceGraphics.py

```
1  import turtle                                                    import turtle
2
3  # Prompt the user for inputting two points
4  x1, y1 = eval(input("Enter x1 and y1 for point 1: "))           enter x1, y1
5  x2, y2 = eval(input("Enter x2 and y2 for point 2: "))           enter x2, y2
6
7  # Compute the distance
8  distance = ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5             compute distance
9
10 # Display two points and the connecting line
11 turtle.penup()
12 turtle.goto(x1, y1) # Move to (x1, y1)                          move to point 1
13 turtle.pendown()
14 turtle.write("Point 1")                                        display point 1
15 turtle.goto(x2, y2) # Draw a line to (x2, y2)                  draw a line
16 turtle.write("Point 2")                                        display point 2
17
18 # Move to the center point of the line
19 turtle.penup()
20 turtle.goto((x1 + x2) / 2, (y1 + y2) / 2)                      move to center
21 turtle.write(distance)                                         display distance
22
23 turtle.done()                                                  pause
```

PYTHON BOOT CAMP

Module 2:
Elementary Programming