2. Simple Python Data

2.1. Variables, Expressions and Statements

2.2. Values and Data Types

2.3. Type conversion functions

2.4. Variables

2.5. Variable Names and Keywords

2.6. Statements and Expressions

2.7. Operators and Operands

2.8. Input

2.9. Order of Operations

2.10. Reassignment

2.10.1. Developing your mental model of How Python Evaluates

2.11. Updating Variables

2.12. Glossary

2.13. Exercises

2.1. Variables, Expressions and Statements

In order to get started learning any programming language there are a number of concepts and ideas that are necessary. The goal of this chapter is to introduce you to the basic vocabulary of programming and some of the fundamental building blocks of Python.

2.2. Values and Data Types

A value is one of the fundamental things — like a word or a number — that a program manipulates. The values we have seen so far are 5 (the result when we added 2 + 3), and "Hello, World!". We often refer to these values as objects and we will use the words value and object interchangeably.

Note

Actually, the 2 and the 3 that are part of the addition above are values(objects) as well.

These objects are classified into different classes, or data types: 4 is an integer, and "Hello, World!" is a string, so-called because it contains a string or sequence of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks.

If you are not sure what class a value falls into, Python has a function called type which can tell you.

1 print(type("Hello, World!"))

2 print(type(17))

3 print("Hello, World")

4

Not surprisingly, strings belong to the class str and integers belong to the class int.

Note

When we show the value of a string using the print function, such as in the third example above, the quotes are not present in the output. The value of the string is the sequence of characters inside the quotes. The quotes are only necessary to help Python know what the value is.

You may have used function notation in a math class, like y = f(x), likely only for functions that act on a single numeric value, and produce a single numeric value. Python has no such restrictions: Inputs and outputs may be of any type.

In the Python shell, it is not necessary to use the print function to see the values shown above. The shell evaluates the Python function and automatically prints the result. For example, consider the shell session shown below. When we ask the shell to evaluate type("Hello, World!"), it responds with the appropriate answer and then goes on to display the prompt for the next use.

>>> type("Hello, World!")

<class 'str'>

>>> type(17)

<class 'int'>

>>> "Hello, World"

'Hello, World'

>>>

Note that in the last example, we simply ask the shell to evaluate the string “Hello, World”. The result is as you might expect, the string itself.

Continuing with our discussion of data types, numbers with a decimal point belong to a class called float, because these numbers are represented in a format called floating-point. At this stage, you can treat the words class and type interchangeably. We’ll come back to a deeper understanding of what a class is in later chapters.

1 print(type(3.2))

2

​

What about values like "17" and "3.2"? They look like numbers, but they are in quotation marks like strings.

1 print(type("17"))

2 print(type("3.2"))

3

​

They’re strings!

Strings in Python can be enclosed in either single quotes (') or double quotes (" - the double quote character), or three of the same separate quote characters (''' or """).

1 print(type('This is a string.') )

2 print(type("And so is this.") )

3 print(type("""and this.""") )

4 print(type('''and even this...''') )

5

Double quoted strings can contain single quotes inside them, as in "Bruce's beard", and single quoted strings can have double quotes inside them, as in 'The knights who say "Ni!"'. Strings enclosed with three occurrences of either quote symbol are called triple quoted strings. They can contain either single or double quotes:

1 print('''"Oh no", she exclaimed, "Ben's bike is broken!"''')

2

Triple quoted strings can even span multiple lines:

1 print("""This message will span

2 several lines

3 of the text.""")

4

Python doesn’t care whether you use single or double quotes or the three-of-a-kind quotes to surround your strings. Once it has parsed the text of your program or command, the way it stores the value is identical in all cases, and the surrounding quotes are not part of the value.

1 print('This is a string.')

2 print("""And so is this.""")

3

So the Python language designers usually chose to surround their strings by single quotes. What do you think would happen if the string already contained single quotes?

When you type a large integer, you might be tempted to use commas between groups of three digits, as in 42,000. This is not a legal integer in Python, but it does mean something else, which is legal:

1 print(42000)

2 print(42,000)

3

​

Well, that’s not what we expected at all! Because of the comma, Python chose to treat this as a pair of values. In fact, the print function can print any number of values as long as you separate them by commas. Notice that the values are separated by spaces when they are displayed.

1 print(42, 17, 56, 34, 11, 4.35, 32)

2 print(3.4, "hello", 45)

3

​

Remember not to put commas or spaces in your integers, no matter how big they are. Also revisit what we said in the previous chapter: formal languages are strict, the notation is concise, and even the smallest change might mean something quite different from what you intended.

Check your understanding

data-2-10: How can you determine the type of a variable?

A. Print out the value and determine the data type based on the value printed.

B. Use the type function.

C. Use it in a known equation and print the result.

D. Look at the declaration of the variable.

data-2-11: What is the data type of ‘this is what kind of data’?

A. Character

B. Integer

C. Float

D. String

2.3. Type conversion functions

Sometimes it is necessary to convert values from one type to another. Python provides a few simple functions that will allow us to do that. The functions int, float and str will (attempt to) convert their arguments into types int, float and str respectively. We call these type conversion functions.

The int function can take a floating point number or a string, and turn it into an int. For floating point numbers, it discards the decimal portion of the number - a process we call truncation towards zero on the number line. Let us see this in action:

1 print(3.14, int(3.14))

2 print(3.9999, int(3.9999)) # This doesn't round to the closest int!

3 print(3.0, int(3.0))

4 print(-3.999, int(-3.999)) # Note that the result is closer to zero

5

​

6 print("2345", int("2345")) # parse a string to produce an int

7 print(17, int(17)) # int even works on integers

8 print(int("23bottles"))

9

The last case shows that a string has to be a syntactically legal number, otherwise you’ll get one of those pesky runtime errors. Modify the example by deleting the bottles and rerun the program. You should see the integer 23.

The type converter float can turn an integer, a float, or a syntactically legal string into a float.

1 print(float("123.45"))

2 print(type(float("123.45")))

3

​

The type converter str turns its argument into a string. Remember that when we print a string, the quotes are removed. However, if we print the type, we can see that it is definitely str.

1 print(str(17))

2 print(str(123.45))

3 print(type(str(123.45)))

4

​

Check your understanding

data-3-4: What value is printed when the following statement executes?

print( int(53.785) )

A. Nothing is printed. It generates a runtime error.

B. 53

C. 54

D. 53.785

2.4. Variables

One of the most powerful features of a programming language is the ability to manipulate variables. A variable is a name that refers to a value.

Assignment statements create new variables and also give them values to refer to.

message = "What's up, Doc?"

n = 17

pi = 3.14159

This example makes three assignments. The first assigns the string value "What's up, Doc?" to a new variable named message. The second gives the integer 17 to n, and the third assigns the floating-point number 3.14159 to a variable called pi.

The assignment token, =, should not be confused with equality (we will see later that equality uses the == token). The assignment statement links a name, on the left hand side of the operator, with a value, on the right hand side. This is why you will get an error if you enter:

17 = n

Tip

When reading or writing code, say to yourself “n is assigned 17” or “n gets the value 17” or “n is a reference to the object 17” or “n refers to the object 17”. Don’t say “n equals 17”.

A common way to represent variables on paper is to write the name with an arrow pointing to the variable’s value. This kind of figure, known as a reference diagram, is often called a state snapshot because it shows what state each of the variables is in at a particular instant in time. (Think of it as the variable’s state of mind). This diagram shows the result of executing the assignment statements shown above.

Reference Diagram

If you ask Python to evaluate a variable, it will produce the value that is currently linked to the variable. In other words, evaluating a variable will give you the value that is referred to by the variable.

1 message = "What's up, Doc?"

2 n = 17

3 pi = 3.14159

4

​

5 print(message)

6 print(n)

7 print(pi)

In each case the result is the value of the variable. To see this in even more detail, we can run the program using codelens.

Variables also have types; again, we can ask the interpreter what they are.

1

message = "What's up, Doc?"

2 n = 17

3 pi = 3.14159

4

​

5 print(type(message))

6 print(type(n))

7 print(type(pi))

8

The type of a variable is the type of the object it currently refers to.

We use variables in a program to “remember” things, like the current score at the football game. But variables are variable. This means they can change over time, just like the scoreboard at a football game. You can assign a value to a variable, and later assign a different value to the same variable.

Note

This is different from math. In math, if you give x the value 3, it cannot change to refer to a different value half-way through your calculations!

A great deal of programming is about having the computer remember things. For example, we might want to keep track of the number of missed calls on your phone. Each time another call is missed, we will arrange to update or change the variable so that it will always reflect the correct value.

Check your understanding

data-4-6: What is printed when the following statements execute?

day = "Thursday"

day = 32.5

day = 19

print(day)

A. Nothing is printed. A runtime error occurs.

B. Thursday

C. 32.5

D. 19

2.5. Variable Names and Keywords

Variable names can be arbitrarily long. They can contain both letters and digits, but they have to begin with a letter or an underscore. Although it is legal to use uppercase letters, by convention we don’t. If you do, remember that case matters. Bruce and bruce are different variables.

Caution

Variable names can never contain spaces.

The underscore character ( \_) can also appear in a name. It is often used in names with multiple words, such as my\_name or price\_of\_tea\_in\_china. There are some situations in which names beginning with an underscore have special meaning, so a safe rule for beginners is to start all names with a letter.

If you give a variable an illegal name, you get a syntax error. In the example below, each of the variable names is illegal.

76trombones = "big parade"

more$ = 1000000

class = "Computer Science 101"

76trombones is illegal because it does not begin with a letter. more$ is illegal because it contains an illegal character, the dollar sign. But what’s wrong with class?

It turns out that class is one of the Python keywords. Keywords define the language’s syntax rules and structure, and they cannot be used as variable names. Python has thirty-something keywords (and every now and again improvements to Python introduce or eliminate one or two):

and

as

assert

break

class

continue

def

del

elif

else

except

exec

finally

for

from

global

if

import

in

is

lambda

nonlocal

not

or

pass

raise

return

try

while

with

yield

True

False

None

You might want to keep this list handy. If the interpreter complains about one of your variable names and you don’t know why, see if it is on this list.

Programmers generally choose names for their variables that are meaningful to the human readers of the program — they help the programmer document, or remember, what the variable is used for.

Caution

Beginners sometimes confuse “meaningful to the human readers” with “meaningful to the computer”. So they’ll wrongly think that because they’ve called some variable average or pi, it will somehow automagically calculate an average, or automagically associate the variable pi with the value 3.14159. No! The computer doesn’t attach semantic meaning to your variable names.

So you’ll find some instructors who deliberately don’t choose meaningful names when they teach beginners — not because they don’t think it is a good habit, but because they’re trying to reinforce the message that you, the programmer, have to write some program code to calculate the average, or you must write an assignment statement to give a variable the value you want it to have.

Check your understanding

data-5-1: True or False: the following is a legal variable name in Python: A\_good\_grade\_is\_A+

A. True

B. False

2.6. Statements and Expressions

A statement is an instruction that the Python interpreter can execute. We have only seen the assignment statement so far. Some other kinds of statements that we’ll see shortly are while statements, for statements, if statements, and import statements. (There are other kinds too!)

An expression is a combination of values, variables, operators, and calls to functions. Expressions need to be evaluated. If you ask Python to print an expression, the interpreter evaluates the expression and displays the result.

1 print(1 + 1)

2 print(len("hello"))

3

In this example len is a built-in Python function that returns the number of characters in a string. We’ve previously seen the print and the type functions, so this is our third example of a function!

The evaluation of an expression produces a value, which is why expressions can appear on the right hand side of assignment statements. A value all by itself is a simple expression, and so is a variable. Evaluating a variable gives the value that the variable refers to.

1 y = 3.14

2 x = len("hello")

3 print(x)

4 print(y)

5

​

If we take a look at this same example in the Python shell, we will see one of the distinct differences between statements and expressions.

>>> y = 3.14

>>> x = len("hello")

>>> print(x)

5

>>> print(y)

3.14

>>> y

3.14

>>>

Note that when we enter the assignment statement, y = 3.14, only the prompt is returned. There is no value. This is due to the fact that statements, such as the assignment statement, do not return a value. They are simply executed.

On the other hand, the result of executing the assignment statement is the creation of a reference from a variable, y, to a value, 3.14. When we execute the print function working on y, we see the value that y is referring to. In fact, evaluating y by itself results in the same response.

2.7. Operators and Operands

Operators are special tokens that represent computations like addition, multiplication and division. The values the operator works on are called operands.

The following are all legal Python expressions whose meaning is more or less clear:

20 + 32

hour - 1

hour \* 60 + minute

minute / 60

5 \*\* 2

(5 + 9) \* (15 - 7)

The tokens +, -, and \*, and the use of parentheses for grouping, mean in Python what they mean in mathematics. The asterisk (\*) is the token for multiplication, and \*\* is the token for exponentiation. Addition, subtraction, multiplication, and exponentiation all do what you expect.

1

print(2 + 3)

2

print(2 - 3)

3

print(2 \* 3)

4

print(2 \*\* 3)

5

print(3 \*\* 2)

6

When a variable name appears in the place of an operand, it is replaced with the value that it refers to before the operation is performed. For example, what if we wanted to convert 645 minutes into hours. In Python 3, division is denoted by the operator token / which always evaluates to a floating point result.

1 minutes = 645

2 hours = minutes / 60

3 print(hours)

4

What if, on the other hand, we had wanted to know how many whole hours there are and how many minutes remain. To help answer this question, Python gives us a second flavor of the division operator. This version, called integer division, uses the token //. It always truncates its result down to the next smallest integer (to the left on the number line).

1 print(7 / 4)

2 print(7 // 4)

3 minutes = 645

4 hours = minutes // 60

5 print(hours)

6

Pay particular attention to the first two examples above. Notice that the result of floating point division is 1.75 but the result of the integer division is simply 1. Take care that you choose the correct flavor of the division operator. If you’re working with expressions where you need floating point values, use the division operator /. If you want an integer result, use //.

The modulus operator, sometimes also called the remainder operator or integer remainder operator works on integers (and integer expressions) and yields the remainder when the first operand is divided by the second. In Python, the modulus operator is a percent sign (%). The syntax is the same as for other operators.

1 quotient = 7 // 3 # This is the integer division operator

2 print(quotient)

3 remainder = 7 % 3

4 print(remainder)

5

​

In the above example, 7 divided by 3 is 2 when we use integer division and there is a remainder of 1 when we use the modulus operator.

The modulus operator turns out to be surprisingly useful. For example, you can check whether one number is divisible by another—if x % y is zero, then x is divisible by y. Also, you can extract the right-most digit or digits from a number. For example, x % 10 yields the right-most digit of x (in base 10). Similarly x % 100 yields the last two digits.

Finally, returning to our time example, the remainder operator is extremely useful for doing conversions, say from seconds, to hours, minutes and seconds.

Check your understanding

data-7-6: What value is printed when the following statement executes?

print(18 / 4)

A. 4.5

B. 5

C. 4

D. 2

data-7-7: What value is printed when the following statement executes?

print(18 // 4)

A. 4.25

B. 5

C. 4

D. 2

data-7-8: What value is printed when the following statement executes?

print(18 % 4)

A. 4.25

B. 5

C. 4

D. 2

2.8. Input

The program in the previous section works fine but is very limited in that it only works with one value for total\_secs. What if we wanted to rewrite the program so that it was more general. One thing we could do is allow the user to enter any value they wish for the number of seconds. The program could then print the proper result for that starting value.

In order to do this, we need a way to get input from the user. Luckily, in Python there is a built-in function to accomplish this task. As you might expect, it is called input.

n = input("Please enter your name: ")

The input function allows the user to provide a prompt string. When the function is evaluated, the prompt is shown. The user of the program can enter the name and press return. When this happens the text that has been entered is returned from the input function, and in this case assigned to the variable n. Make sure you run this example a number of times and try some different names in the input box that appears.

1 n = input("Please enter your name: ")

2 print("Hello", n)

3

It is very important to note that the input function returns a string value. Even if you asked the user to enter their age, you would get back a string like "17". It would be your job, as the programmer, to convert that string into an int or a float, using the int or float converter functions we saw earlier.

To modify our previous program, we will add an input statement to allow the user to enter the number of seconds. Then we will convert that string to an integer. From there the process is the same as before. To complete the example, we will print some appropriate output.

1 str\_seconds = input("Please enter the number of seconds you wish to convert")

2 total\_secs = int(str\_seconds)

3​

4 hours = total\_secs // 3600

5 secs\_still\_remaining = total\_secs % 3600

6 minutes = secs\_still\_remaining // 60

7 secs\_finally\_remaining = secs\_still\_remaining % 60

8

​

9 print("Hrs=", hours, "mins=", minutes, "secs=", secs\_finally\_remaining)

10

​

The variable str\_seconds will refer to the string that is entered by the user. As we said above, even though this string may be 7684, it is still a string and not a number. To convert it to an integer, we use the int function. The result is referred to by total\_secs. Now, each time you run the program, you can enter a new value for the number of seconds to be converted.

Check your understanding

data-8-4: What is printed when the following statements execute?

n = input("Please enter your age: ")

# user types in 18

print ( type(n) )

A. <class 'str'>

B. <class 'int'>

C. <class 18>

D. 18

data-8-5: Click on all of the variables of type `int` in the code below

seconds = input("Please enter the number of seconds you wish to convert")

hours = int(seconds) // 3600

total\_secs = int(seconds)

secs\_still\_remaining = total\_secs % 3600

print(secs\_still\_remaining)

data-8-6: Click on all of the variables of type `str` in the code below

seconds = input("Please enter the number of seconds you wish to convert")

hours = int(seconds) // 3600

total\_secs = int(seconds)

secs\_still\_remaining = total\_secs % 3600

print(secs\_still\_remaining)

2.9. Order of Operations

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence. Python follows the same precedence rules for its mathematical operators that mathematics does.

Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 \* (3-1) is 4, and (1+1)\*\*(5-2) is 8. You can also use parentheses to make an expression easier to read, as in (minute \* 100) / 60, even though it doesn’t change the result.

Exponentiation has the next highest precedence, so 2\*\*1+1 is 3 and not 4, and 3\*1\*\*3 is 3 and not 27. Can you explain why?

Multiplication and both division operators have the same precedence, which is higher than addition and subtraction, which also have the same precedence. So 2\*3-1 yields 5 rather than 4, and 5-2\*2 is 1, not 6.

Operators with the same precedence (except for \*\*) are evaluated from left-to-right. In algebra we say they are left-associative. So in the expression 6-3+2, the subtraction happens first, yielding 3. We then add 2 to get the result 5. If the operations had been evaluated from right to left, the result would have been 6-(3+2), which is 1.

Note

An exception to the left-to-right left-associative rule is the exponentiation operator \*\*. A useful hint is to always use parentheses to force exactly the order you want when exponentiation is involved:

1 print(2 \*\* 3 \*\* 2) # the right-most \*\* operator gets done first!

2 print((2 \*\* 3) \*\* 2) # use parentheses to force the order you want!

3

See Operator precedence table for all the operators introduced in this book. You will also see many upcoming non-mathematical Python operators.

Check your understanding

data-9-3: What is the value of the following expression:

16 - 2 \* 5 // 3 + 1

A. 14

B. 24

C. 3

D. 13.667

data-9-4: What is the value of the following expression:

2 \*\* 2 \*\* 3 \* 3

A. 768

B. 128

C. 12

D. 256

2.10. Reassignment

As we have mentioned previously, it is legal to make more than one assignment to the same variable. A new assignment makes an existing variable refer to a new value (and stop referring to the old value).

1 bruce = 5

2 print(bruce)

3 bruce = 7

4 print(bruce)

5

The first time bruce is printed, its value is 5, and the second time, its value is 7. The assignment statement changes the value (the object) that bruce refers to.

It is important to note that in mathematics, a statement of equality is always true. If a is equal to b now, then a will always equal to b. In Python, an assignment statement can make two variables refer to the same object and therefore have the same value. They appear to be equal. However, because of the possibility of reassignment, they don’t have to stay that way:

1 a = 5

2 b = a # after executing this line, a and b are now equal

3 print(a, b)

4 a = 3 # after executing this line, a and b are no longer equal

5pr int(a, b)

6

​

Line 4 changes the value of a but does not change the value of b, so they are no longer equal. We will have much more to say about equality in a later chapter.

2.10.1. Developing your mental model of How Python Evaluates

Its important to start to develop a good mental model of the steps Python takes when evaluating an assignment statement. In an assignment statement Python first evaluates the code on the right hand side of the assignment operator. It then gives a name to whatever that is. The (very short) visualization below shows what is happening.

a = 5

b = a

a = 5

In the first statement a = 5 the literal number 5 evaluates to 5, and is given the name a. In the second statement, the variable a evaluates to 5 and so 5 now ends up with a second name b.

Note

In some programming languages, a different symbol is used for assignment, such as <- or :=. The intent is that this will help to avoid confusion. Python chose to use the tokens = for assignment, and == for equality. This is a popular choice also found in languages like C, C++, Java, and C#.

Check your understanding

data-10-5: After the following statements, what are the values of x and y?

x = 15

y = x

x = 22

A. x is 15 and y is 15

B. x is 22 and y is 22

C. x is 15 and y is 22

D. x is 22 and y is 15

2.11. Updating Variables

One of the most common forms of reassignment is an update where the new value of the variable depends on the old. For example,

x = x + 1

This means get the current value of x, add one, and then update x with the new value. The new value of x is the old value of x plus 1. Although this assignment statement may look a bit strange, remember that executing an assignment is a two-step process. First, evaluate the right-hand side expression. Second, let the variable name on the left-hand side refer to this new resulting object. The fact that x appears on both sides does not matter. The semantics of the assignment statement makes sure that there is no confusion as to the result. The visualizer makes this very clear.

x = 6

x = x + 1

x = x + 1

1 x = 6 # initialize x

2 print(x)

3 x = x + 1 # update x

4 print(x)

5

​

If you try to update a variable that doesn’t exist, you get an error because Python evaluates the expression on the right side of the assignment operator before it assigns the resulting value to the name on the left. Before you can update a variable, you have to initialize it, usually with a simple assignment. In the above example, x was initialized to 6.

Updating a variable by adding 1 is called an increment; subtracting 1 is called a decrement. Sometimes programmers also talk about bumping a variable, which means the same as incrementing it by 1.

Check your understanding

data-11-4: What is printed when the following statements execute?

x = 12

x = x - 1

print(x)

A. 12

B. -1

C. 11

D. Nothing. An error occurs because x can never be equal to x - 1.

data-11-5: What is printed when the following statements execute?

x = 12

x = x - 3

x = x + 5

x = x + 1

print(x)

A. 12

B. 9

C. 15

D. Nothing. An error occurs because x cannot be used that many times in assignment statements.

2.12. Glossary

assignment statement

A statement that assigns a value to a name (variable). To the left of the assignment operator, =, is a name. To the right of the assignment token is an expression which is evaluated by the Python interpreter and then assigned to the name. The difference between the left and right hand sides of the assignment statement is often confusing to new programmers. In the following assignment:

n = n + 1

n plays a very different role on each side of the =. On the right it is a value and makes up part of the expression which will be evaluated by the Python interpreter before assigning it to the name on the left.

assignment token

= is Python’s assignment token, which should not be confused with the mathematical comparison operator using the same symbol.

class

see data type below

comment

Information in a program that is meant for other programmers (or anyone reading the source code) and has no effect on the execution of the program.

data type

A set of values. The type of a value determines how it can be used in expressions. So far, the types you have seen are integers (int), floating-point numbers (float), and strings (str).

decrement

Decrease by 1.

evaluate

To simplify an expression by performing the operations in order to yield a single value.

expression

A combination of operators and operands (variables and values) that represents a single result value. Expressions are evaluated to give that result.

float

A Python data type which stores floating-point numbers. Floating-point numbers are stored internally in two parts: a base and an exponent. When printed in the standard format, they look like decimal numbers. Beware of rounding errors when you use floats, and remember that they are only approximate values.

increment

Both as a noun and as a verb, increment means to increase by 1.

initialization (of a variable)

To initialize a variable is to give it an initial value. Since in Python variables don’t exist until they are assigned values, they are initialized when they are created. In other programming languages this is not the case, and variables can be created without being initialized, in which case they have either default or garbage values.

int

A Python data type that holds positive and negative whole numbers.

integer division

An operation that divides one integer by another and yields an integer. Integer division yields only the whole number of times that the numerator is divisible by the denominator and discards any remainder.

keyword

A reserved word that is used by the compiler to parse program; you cannot use keywords like if, def, and while as variable names.

modulus operator

Also called remainder operator or integer remainder operator. Gives the remainder after performing integer division.

object

Also known as a data object (or data value). The fundamental things that programs are designed to manipulate (or that programmers ask to do things for them).

operand

One of the values on which an operator operates.

operator

A special symbol that represents a simple computation like addition, multiplication, or string concatenation.

prompt string

Used during interactive input to provide the user with hints as to what type of value to enter.

reference diagram

A picture showing a variable with an arrow pointing to the value (object) that the variable refers to. See also state snapshot.

rules of precedence

The set of rules governing the order in which expressions involving multiple operators and operands are evaluated.

state snapshot

A graphical representation of a set of variables and the values to which they refer, taken at a particular instant during the program’s execution.

statement

An instruction that the Python interpreter can execute. So far we have only seen the assignment statement, but we will soon meet the import statement and the for statement.

str

A Python data type that holds a string of characters.

type conversion function

A function that can convert a data value from one type to another.

value

A number or string (or other things to be named later) that can be stored in a variable or computed in an expression.

variable

A name that refers to a value.

variable name

A name given to a variable. Variable names in Python consist of a sequence of letters (a..z, A..Z, and \_) and digits (0..9) that begins with a letter. In best programming practice, variable names should be chosen so that they describe their use in the program, making the program self documenting.

2.13. Exercises

Evaluate the following numerical expressions in your head, then use the active code window to check your results:

5 \*\* 2

9 \* 5

15 / 12

12 / 15

15 // 12

12 // 15

5 % 2

9 % 5

15 % 12

12 % 15

6 % 6

0 % 7

Many people keep time using a 24 hour clock (11 is 11am and 23 is 11pm, 0 is midnight). If it is currently 13 and you set your alarm to go off in 50 hours, it will be 15 (3pm). Write a Python program to solve the general version of the above problem. Ask the user for the time now (in hours), and then ask for the number of hours to wait for the alarm. Your program should output what the time will be on the clock when the alarm goes off.

It is possible to name the days 0 through 6 where day 0 is Sunday and day 6 is Saturday. If you go on a wonderful holiday leaving on day number 3 (a Wednesday) and you return home after 10 nights you would return home on a Saturday (day 6) Write a general version of the program which asks for the starting day number, and the length of your stay, and it will tell you the number of day of the week you will return on.

Take the sentence: All work and no play makes Jack a dull boy. Store each word in a separate variable, then print out the sentence on one line using print.

Add parenthesis to the expression 6 \* 1 - 2 to change its value from 4 to -6.

formula for compound interest

Write a Python program that assigns the principal amount of 10000 to variable P, assign to n the value 12, and assign to r the interest rate of 8% (0.08). Then have the program prompt the user for the number of years, t, that the money will be compounded for. Calculate and print the final amount after t years.

Write a program that will compute the area of a circle. Prompt the user to enter the radius and print a nice message back to the user with the answer.

Write a program that will compute the area of a rectangle. Prompt the user to enter the width and height of the rectangle. Print a nice message with the answer.

Write a program that will compute MPG for a car. Prompt the user to enter the number of miles driven and the number of gallons used. Print a nice message with the answer.

Write a program that will convert degrees celsius to degrees fahrenheit.

Write a program that will convert degrees fahrenheit to degrees celsius.