**Variables, expressions and statements**

**Values and data types**

A **value** is one of the fundamental things — like a letter or a number — that a program manipulates. The values we have seen so far are 4(as an integer) (the result when we added 2 + 2), and "Hello, World!"(as a string).

These values are classified into different classes, or data types:

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

|  |
| --- |
| >>> type("Hello, world")  <class 'str'>  >>> type( 2 + 2)  <class 'int'>  >>> type(2.5)  <class 'float'>  >>> type(['amit', 'mukul', 'singh','rajput'])  <class 'list'>  >>> type("17")  <class 'str'>  >>> type("3.2")  <class 'str'> |

Strings in Python can be enclosed in either single quotes (') or double quotes ("), or three of each (''' or """)

Strings enclosed with three occurrences of either quote symbol are called triple quoted strings.

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:

|  |
| --- |
| >>> 42000  42000  >>> 42,000  (42, 0) |

**Variables**

A variable is a name that refers to a value.

The assignment statement gives a value to a variable:

|  |
| --- |
| >>> message = "What's up, Doc?"  >>> n = 17  >>> pi = 3.14159 |

The assignment token, =, should not be confused with equals, which uses the token ==. The assignment statement binds a name, on the left-hand side of the operator, to a value, on the right-hand side.

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

**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. ***Amit***and ***amit*** are different variables.

The underscore character ( \_) can appear in a name. It is often used in names with multiple words, such as my\_name or price\_of\_tea\_in\_india.

Keywords define the language’s syntax rules and structure,

and they cannot be used as variable names.

**Statements**

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!)

**Evaluating expressions**

An expression is a combination of values, variables, operators, and calls to functions. If you type an expression at the

Python prompt, the interpreter evaluates it and displays the result:

|  |
| --- |
| >>> 1 + 1  2  >>> len("hello")  5 |

**Operators and operands**

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

The following are all legal Python expressions.

|  |
| --- |
| 20+32 hour-1 hour\*60+minute minute/60 5\*\*2 (5+9)\*(15-7) |

The tokens +, -, and \* , and the use of parenthesis for grouping, mean in Python what they mean in mathematics. The asterisk ( \* ) is the token for multiplication, and \*\* is the token for exponentiation.

|  |
| --- |
| >>> 2 \*\* 3  8  >>> 3 \*\* 2  9 |

Oops! In Python 3, the division operator / always yields a floating point result. What we might have wanted to know was how many whole hours there are, and how many minutes remain. Python gives us two different flavors of the division operator. The second, called floor division uses the token //. Its result is always a whole number — and if it has to adjust the number it always moves it to the left on the number line. So 6 // 4 yields 1, but -6 // 4 might surprise you!

|  |
| --- |
| >>> 7 / 4  1.75  >>> 7 // 4  1  >>> minutes = 645  >>> hours = minutes // 60  >>> hours  10 |

**Type converter functions**

Here we’ll look at three more Python functions, int, float and str, which will (attempt to) convert their arguments into types int, float and str respectively. We call these type converter functions.

|  |
| --- |
| >>> int(3.14)  3  >>> float(17)  17.0  >>> str(17)  '17' |

**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. The acronym **PEMDAS** is a useful way to remember the order of operations:

1. Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you

want.

2. Exponentiation has the next highest precedence,

3. Multiplication and both Division operators have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence.

4. Operators with the same precedence are evaluated from left-to-right. In algebra we say they are left-associative.

*Due to some historical quirk, an exception to the left-to-right left-associative rule is the exponentiation operator \*\* , so a useful hint is to always use parentheses to force exactly the order you want when exponentiation is involved:*

|  |
| --- |
| >>> 2 \*\* 3 \*\* 2 # The right-most \*\* operator gets done first!  512  >>> (2 \*\* 3) \*\* 2 # Use parentheses to force the order you want!  64 |

**Operations on strings**

In general, you cannot perform mathematical operations on strings, even if the strings look like numbers. The following are illegal (assuming that message has type string):

|  |
| --- |
| >>> message – 1 #Error  >>> "Hello" / 123 #Error  >>> message \* "Hello" #Error  >>> "15" + 2 #Error |

Interestingly, the + operator does work with strings, but for strings, the + operator represents concatenation, not addition. Concatenation means joining the two operands by linking them end-to-end.

|  |
| --- |
| >>>fruit = "banana"  >>>baked\_good = " nut bread"  >>>print(fruit + baked\_good)  banana nut bread |

The \* operator also works on strings; it performs repetition. For example, 'Fun'\*3 is 'FunFunFun'. One of the operands has to be a string; the other has to be an integer.

**Input**

There is a built-in function in Python for getting input from the user:

|  |
| --- |
| name = input("Please enter your name: ") |

The user of the program can enter the name and click OK, and when this happens the text that has been entered is returned from the input function, and in this case assigned to the variable name.

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 a int or a float, using the int or float converter functions.

**Composition**

One of the most useful features of programming languages is their ability to take small building blocks and compose them into larger chunks.

Ex→ Asks the user to input a value for the radius of a circle, and then computes the area of the circle from the formula Area = πR \*\*2.

Firstly, we’ll do the four steps one at a time:

|  |
| --- |
| response = input("What is your radius? ")  r = float(response)  area = 3.14159 \* r\*\*2  print("The area is ", area) |

Now let’s compose the first two lines into a single line of code, and compose the second two lines into another line of

code.

|  |
| --- |
| r = float( input("What is your radius? ") )  print("The area is ", 3.14159 \* r\*\*2) |

If we really wanted to be tricky, we could write it all in one statement:

|  |
| --- |
| print("The area is ", 3.14159\*float(input("What is your radius?"))\*\*2) |

**The modulus operator**

The **modulus operator** works on integers (and integer expressions) and gives the remainder when the first number is divided by the second. In Python, the modulus operator is a percent sign (%).It has the same precedence as the multiplication operator.

|  |
| --- |
| >>> q = 7 // 3 # This is integer division operator  >>> print(q)  2  >>> r = 7 % 3  >>> print(r)  1 |

***write a program to ask the user to enter some seconds, and we’ll convert them into hours, minutes, and remaining seconds.***

|  |
| --- |
| total\_secs = int(input("How many seconds, in total?"))  hours = total\_secs // 3600  secs\_still\_remaining = total\_secs % 3600  minutes = secs\_still\_remaining // 60  secs\_finally\_remaining = secs\_still\_remaining % 60  print("Hrs=", hours, "mins=", minutes, "secs=", secs\_finally\_remaining) |