Expressions and Variables

What are Expressions?

- We said that the syntax of the print statement was `print(`*`expression`*`)` - but what do we mean by 'expression'?
- An expression is a piece of code that the interpreter can *evaluate* to produce a *value*.
- For example, a mathematical calculation might be evaluated to give a certain number.
- Expressions are an integral part of most programming languages.
- Let's see what we can do with them…

Data Types

- An expression might evaluate to one of several *data types* - i.e. kinds of data.
- We will see many, but for now, we will only care about:
    - Integers: 1, 2, 3, 42, 100, -5, etc.
    - Floating Points (floats): 2.5, 7.0, 3.14159, etc.
        - Note that 7.0 is a float, vs 7 the integer.
    - Strings (text): "Hello World", "Learn to Code", "100", etc.
        - Note that "100" is a string, despite only containing a number. We will come back to why you might want to make that distinction later.
    - Boolean: True / False.
        - We will come back to these later on.
- Why do we make this distinction?
    - Different types of data can be treated differently.
    - Some operations do not make sense on certain data types. For example, what does it mean to take the square root of a string?!
- Many languages are *statically typed*, and will force you to be explicit about what type of data you are working with.
- Python is *dynamically typed* - it doesn't care!

Integer and Floating Point Expressions

- Let's keep things simple, and start with numbers.
- There are some subtle distinctions between integers and floats, but we shall use them interchangeably for the time being.
- Remember, we can use `print` to output our expressions!
    - On the interactive prompt, we can also just type our expressions on their own, and the interpreter will evaluate them for us.
    - In our programs though, we will need to use `print`!
- All numbers are, on their own, expressions.
    - `print(42)`
- We can do maths.
    - `print(2+2)`
- Available maths operations (go through each one):
    - `+`
    - `-`
    - `*`
    - `/`

    - `**` Raise to power
    - `%` Modulo - gives remainder
    - `()` Brackets
- Maths is evaluated in order of precedence - BIDMAS / BODMAS.
    - Refresh on this if people don’t know it.
    - Can use brackets to make things evaluate in desired order.
        - `2 * 4 + 3 * 3 = 8 + 9 = 17`
        - vs
        - `2 * (4 + 3) * 3 = 2 * 7 * 3 = 42`
- We can use Python like a calculator! Computers are good at calculating…
    - Have students try out mathematical expressions (and `print`) by asking them to write a program to calculate:
    - $\frac{-6+\sqrt{6^2-4*3*-15}}{2*3}$
    - Hint: square root is the same as raising to power of 0.5.
    - Valid expressions (there may be other variants):
        - `(-6 + (6*6 - 4*3*-15) ** 0.5) / (2*3)`
        - `(-6 + (6**2 - 4*3*-15) ** 0.5) / (2*3)`
    - Correct answer:
        - `1.449489742783178`
    - Note the effects of operator precedence.
    - Note also that spacing within expression does not change output of expression, but can improve readability.

String Expressions

- Strings are another data type - they represent text, i.e. ‘strings of characters’.
- Strings are simple: just put some text inside a pair of quotes!
    - We saw a string expression earlier: `“Hello World!”`.
    - In Python, you can use single or double quotes (or triple quotes). As a stylistic recommendation, we suggest:
        - Double quotes for strings. `“Learn to Code”`
        - Single quotes for single characters. `‘f’`
    - ...but it doesn’t matter too much.
- **Watch out!** Word processors (such as what we made these notes in…) will usually use angled quotes to make them prettier. Python will not understand these! Use ‘normal’ quotes.
    - e.g. Bad: `“Hello World!”`
    - Good: `"Hello World!"`
- What operations can we perform on strings?
    - There are a lot, but we will just look at a few.
    - We might come back to the fancier stuff later on (namely once we have covered classes / objects and their syntax).
- Concatenation:
    - `“Hello” + “ ” + “World!”`
    - `= “Hello World!”`
    - Why is this useful, when we could just do it all as one string? We will see why later!

- Repetition:
    - `“badger” * 3`
    - `= “badgerbadgerbadger”`
    - (This is a bit of an abuse of notation, personally...)
- Note that we cannot use subtraction or division - these produce errors!
    - Demonstrate.
    - These operations do not make sense on strings!
- Note also that strings and numbers are different data types. We cannot add numbers to strings.
    - Demonstrate: `“I am “ + (10 + 3) + “ years old.”` will not work.
    - Note the error message. It tells you what the interpreter could not understand.
- If we want to treat numbers like text, we can put them in quotes to treat them as strings, but this does not let us do calculations.
    - Demonstrate: `“I am “ + “(10 + 3)” + “ years old.”` does not give the desired output.
- We can use string formatting!
    - `“I am %d years old.” % (10 + 3)`
    - *Format specifiers* such as %d are replaced with values listed after % symbol.
    - What is happening here?
        - 10 + 3 is evaluated to the integer 13.
        - 13 is then converted into a string “13”.
        - “%d” is replaced with “13”.
- Format specifiers (taken from C - we only list a few common ones here) should match data type:
    - %d or %i - Integer. (well, a signed integer)
    - %f - Floating point.
    - %e - Floating point in scientific notation.
    - %s - String.
- List of values should be comma-separated list in brackets.
    - `“My name is %s, I am %d years old, and I have %d %s.” % (“Alex”, 13, 7, “friends”)`
- Simple exercises:
    - Adjust the above expression for yourself by adjusting the parameters.
    - Work out how to change the above so we can adjust the units of time from ‘years’ to ‘months’.
        - `“My name is %s, I am %d %s old, and I have %d %s.” % (“Alex”, 13*12, “months”, 7, “friends”)`

Variables

- So far we have been hard-coding all of these expressions.
- A valid question: We could have just calculated these ahead of time! What is the the point of expressions?
- Answer: variables.
    - We want to store the results of our calculations somewhere, for later use.
    - We also want a way to remember information a user has given.
    - We might want to use these values for further calculations.

- A *variable* is a little piece of memory that we can give a name to and set aside for storing a value.
- We can define as many variables as we have memory for (which, on modern computers, will be a lot!).
- We define a variable as follows:
    - `variablename = expression`
    - e.g:
    - `x = 10 * 8 + 5`
    - `age = 32`
    - `firstname = "Alice"`
- Variable names can contain letters, numbers, underscores, but cannot start with numbers.
    - Variable names are case sensitive!
        - `age` and `Age` are two different variables.
    - Variable names cannot* be the name of a Python command. For example, you cannot have a variable called print!
        - *Except you sort of can, because Python is weird like that.
        - Function names are best avoided, basically.
- We must define a variable before we can use it.

    -
      ```
      print(x)
      x = 10
      # error!
      ```
- We can change the value of a variable with the same syntax.

    -
      ```
      x = 10
      print(x)
      # outputs 10
      x = 4
      print(x)
      #outputs 4
      ```
- Because Python is dynamically-typed, we can also re-assign a variable with a value of a different data type.
    - This cannot be done in statically-typed languages.
    - We recommend you avoid doing this! You risk causing bugs in your program if you do.
- Exercise:
    - Recall from earlier:
        - `"My name is %s, I am %d years old, and I have %d %s." % ("Alex", 13, 7, "friends")`
    - Create variables for each of the four values, with appropriate names, and print the string using the values from the variables.
    - Now add a second print statement, that outputs the same information in a different order. For example "I am a 13 year old with 7 friends, called Alex.".
    - Try running the program several times, with different values in the variables, and observe how the output changes.

Storing User Input in Variables

- Any useful program is going to need to take input from the user!
- We will want to store the input in a variable so we can use it.
- The Python function `input` allows us to do this. It does the following:
    - Displays a message prompt to the user.
    - Waits for the user to type some input and press enter.
    - Returns the *value* that was input.
- Since a value is returned, we can use it in an expression, and assign it to a variable.
    - Example: `animal = input(“Please enter your favourite animal: ”)`
- Exercise:
    - Write a program that asks the user for their name, then prints a greeting that uses their name.
    - Some possible solutions:

        -
          ```
          name = input(“What is your name? ”)
          print(“Hello, %s.” % name)
          ```

        -
          ```
          name = input(“What is your name? ”)
          print(“Hello, ” + name + “.”)
          ```

        -
          ```
          print(“Hello, %s.” % input(“What is your name? ”))
          ```
    - Discuss: which of these is better?
        - First is probably neater than second.
        - The third one is quite messy, but it does work.
            - Make sure people understand why it works.
        - Readable code is better than fewer lines of code!
- Exercise with a trick:
    - Write a program that asks for two numbers, then prints their sum.
    - Before starting exercise, ask if anyone can see what might go wrong with this!
        - Input command inputs strings, not numbers, so adding values will concatenate strings, rather than sum numbers.
    - Get students to try it anyway.
    - Code:

        -
          ```
          a = input(“Enter a number: ”)
          b = input(“And another: ”)
          print(a + b)
          ```
    - Inputting 2 and 3 will output 23!
- We need to convert any data type into another data type. There are functions to do this:
    - `int()`
    - `float()`
    - `str()`
- Note that these will give an error if the input doesn’t make sense!
    - Example: `int(“hello”)`
- Exercise: fix the previous exercise! Two possible answers:

    -
      ```
      a = int(input(“Enter a number: ”))
      b = int(input(“And another: ”))
      print(a + b)
      ```

  -
    ```
    a = input(“Enter a number: “)
    b = input(“And another: “)
    print(int(a) + int(b))
    ```
  - Make sure people understand why these work. The key is that we can build expressions out of other expressions!

Final Exercise

- Write a program that inputs the radius of a circle, and outputs the circumference and area of the circle.
  - Bonus points for formatted output rather than simply outputting numbers.
  - Maths needed:
    - π = 3.14159
    - circumference = 2πr
    - area = πr^2
  - Sample answer:

    -
      ```
      r = float(input(“Radius: “))
      pi = 3.14159
      circ = 2 * pi * r
      area = pi * r * r
      print(“Circumference: %f” % circ)
      print(“Area: %f” % area)
      ```
  - Discussion points:
    - Need to convert input away from string, and to a *float* rather than an *int*. Similarly, need to use floats in output.
    - Better to code a variable for pi rather than hard-code the value in both equations. Reusing constant values is good practice - if we need to change the precision (or value!!!) of pi, we only have to change one part of the code.
  - Extension if there is spare time:
    - Write similar programs for other shapes! Look up the maths and implement. Ellipse, cube, etc.

Summary

- We have seen:
  - How to output information to the terminal.
  - Expressions, and how we can build expressions that perform calculations and build strings.
  - Variables, and how we can use them in expressions.
  - How to take and store user input from the terminal to use in our variables and expressions.
- This is enough to write programs that perform simple, calculation-based processing.
- We don’t yet know how to have our programs make decisions based on these calculations.
- ...so, up next: making decisions!

Next lesson: Making Decisions