Building Python Programs

Chapter 2: Data and Definite Loops

Data and expressions

Data types

- Internally, computers store everything as 1s and 0s

```
104   → 01101000
'hi'  → 0110100001101001
'h'   → 01101000
```

- How are `h` and `104` differentiated?

- **type**: A category or set of data values.
  - Constrains the operations that can be performed on data
  - Many languages ask the programmer to specify types
  - Examples: integer, real number, string

Python's number types

Name	Description	Examples
`int`	integers	`42, -3, 0, 926394`
`float`	real numbers	`3.1, -0.25`
`complex`		

Expressions

- **expression**: A value or operation that computes a value.
  - Examples:
    ```
    1 + 4 * 5
    (7 + 2) * 6 / 3
    42.0
    ```
  - The simplest expression is a *literal value*.
  - A complex expression can use operators and parentheses.

Arithmetic operators

- **operator**: Combines multiple values or expressions.

`+`	addition
–	subtraction (or negation)
`*`	multiplication
`/`	division
//	integer division (a.k.a. leave off any remainder)
`%`	modulus (a.k.a. remainder)
**	exponent

- As a program runs, its expressions are *evaluated*.
  - `1 + 1` evaluates to `2`

Integer division with //

- When we divide integers with //, the quotient is also an integer.
  - `14 // 4` is `3`, not `3.5`

```
         3               4                 52
      ____            ____           ________
 4 )  14        10 )  45        27 )  1425
      12              40              135
      --              --              ---
       2               5               75
                                       54
                                       --
                                       21
```

- More examples:
  - `32 // 5` is `6`
  - `84 // 10` is `8`
  - `156 // 100` is `1`

  - Dividing by 0 causes an error when your program runs.

Integer remainder with `%`

- The `%` operator computes the remainder from integer division.
  - `14 % 4` is `2`
  - `218 % 5` is `3`

```
        3              43
  4 ) 14         5 ) 218
      12             20
       2              18
                      15
                       3
```

What is the result?

```
45 % 6
2 % 2
8 % 20
11 % 0
```

- Applications of `%` operator:
  - Obtain last digit of a number: `230857 % 10` is `7`
  - Obtain last 4 digits: `658236489 % 10000` is `6489`
  - See whether a number is odd: `7 % 2` is `1`, `42 % 2` is `0`

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.

    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `*` `/` `//` `%` have a higher level of precedence than `+` `-`

    ```
    1 + 3 * 4                 is 13

    6 + 8 // 2 * 3
    6 +   4    * 3
    6 +     12                is 18
    ```

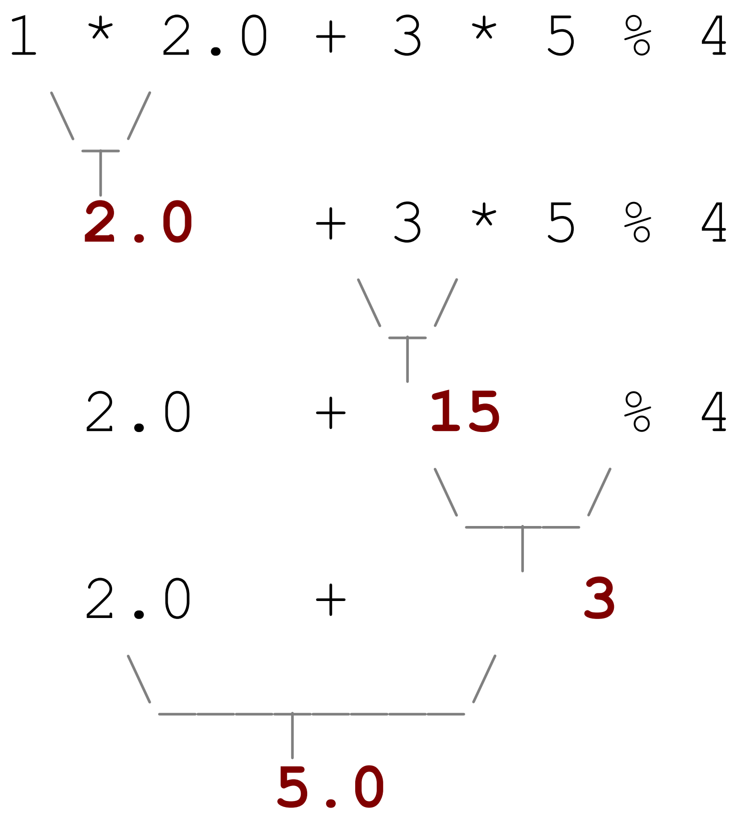
  - Parentheses can force a certain order of evaluation:

    ```
    (1 + 3) * 4               is 16
    ```

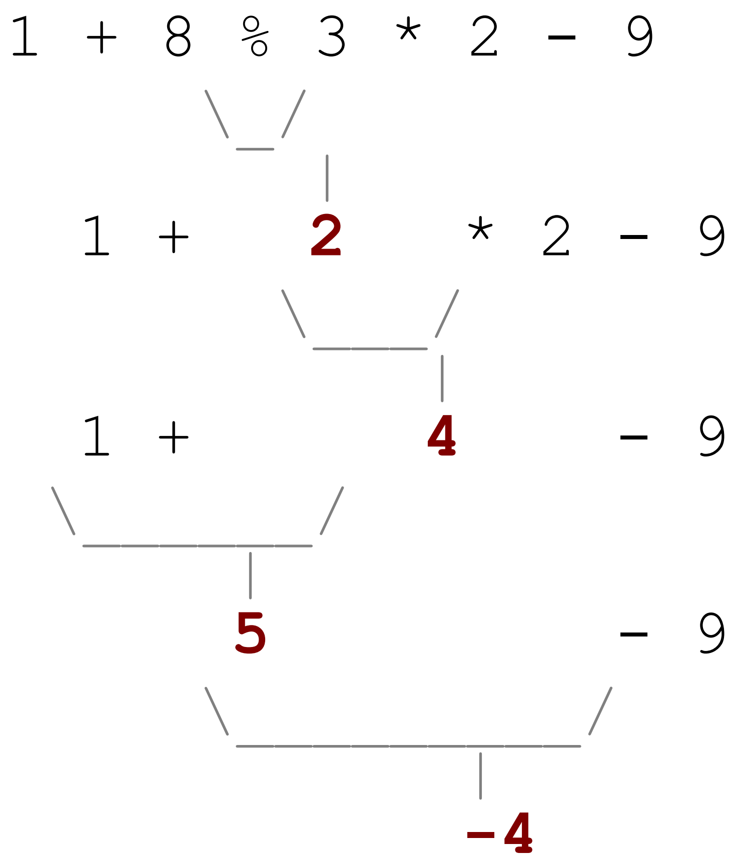
  - Spacing does not affect order of evaluation

    ```
    1+3 * 4-2                 is 11
    ```

Precedence examples



```
1 * 2.0 + 3 * 5 % 4
2.0     + 3 * 5 % 4
2.0     + 15    % 4
2.0     +       3
5.0
```



```
1 + 8 % 3 * 2 - 9
1 + 2 * 2 - 9
1 + 4 - 9
5 - 9
-4
```

Precedence questions

- What values result from the following expressions?
  - `9 // 5`
  - `695 % 20`
  - `7 + 6 * 5`
  - `7 * 6 + 5`
  - `248 % 100 / 5`
  - `6 * 3 - 9 // 4`
  - `(5 - 7) * 2 ** 2`
  - `6 + (18 % (17 - 12))`

Variables

Receipt example

What's bad about the following code?

```
# Calculate total owed, assuming 8% tax / 15% tip
print("Subtotal:")
print(38 + 40 + 30)

print("Tax:")
print((38 + 40 + 30) * .08)

print("Tip:")
print((38 + 40 + 30) * .15)

print("Total:")
print(38 + 40 + 30 + (38 + 40 + 30) * .15 + (38 + 40 + 30) * .08)
```

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `print` statements

Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:

  - Steps for using a variable:
    - *Declare/initialize* it - state its name and type and store a value into it
    - *Use* it - print it or use it as part of an expression

Declaration and assignment

- **variable declaration and assignment**:

  Sets aside memory for storing a value and stores a value into a variable.

  - Variables must be declared before they can be used.
  - The value can be an expression; the variable stores its result.

- Syntax:

  name = **expression**

  - **zipcode = 90210**
  - **myGPA = 1.0 + 2.25**

zipcode	90210

myGPA	3.25

Using variables

- Once given a value, a variable can be used in expressions:

```
x = 3            # x is 3
y = 5 * x - 1    # now y is 14
```

- You can assign a value more than once:

```
x = 3            # 3 here


x = 4 + 7        # now x is 11
```

x	11

Assignment and algebra

- Assignment uses = , but it is not an algebraic equation.
    - = means, *"store the value at right in variable at left"*
        - The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?

```
x = 3
x = x + 2    # ???
```

x	5

Receipt question

Improve the receipt program using variables.

```
def main():
    # Calculate total owed, assuming 8% tax / 15% tip
    print("Subtotal:")
    print(38 + 40 + 30)

    print("Tax:")
    print((38 + 40 + 30) * .08)

    print("Tip:")
    print((38 + 40 + 30) * .15)

    print("Total:")
    print(38 + 40 + 30 + (38 + 40 + 30) * .15 + (38 + 40 + 30) * .08)
```

Printing a variable's value

- Use a comma to print a string and a variable's value on one line.

  -
    ```
    grade = (95.1 + 71.9 + 82.6) / 3.0
    print("Your grade was", grade)

    students = 11 + 17 + 4 + 19 + 14
    print("There are", students,
           "students in the course.")
    ```
  - Output:

    ```
    Your grade was 83.2
    There are 65 students in the course.
    ```

Receipt answer

```
def main():
    # Calculate total owed, assuming 8% tax / 15% tip
    subtotal = 38 + 40 + 30        # int
    tax = subtotal * .08           # float
    tip = subtotal * .15           # float
    total = subtotal + tax + tip   # float

    print("Subtotal:", subtotal)
    print("Tax:", tax)
    print("Tip:", tip)
    print("Total:", total)
```

for loops

Getting rid of repetition

- Functions

- Variables

- String Multiplication
  - Allows you to print multiple occurrences of the same string without typing them all out

```
print("meow" * 3)              # meowmeowmeow
```

- What if you want to repeat function calls?

Repetition with `for` loops

- So far, repeating an action results in redundant code:

```
make_batter()
bake_cookies()
bake_cookies()
bake_cookies()
bake_cookies()
bake_cookies()
frost_cookies()
```

- Python's **`for` loop** statement performs a task many times.

```
mix_batter()
for i in range(1, 6):     # repeat 5 times
    bake_cookies()
frost_cookies()
```

for loop syntax

```
for variable in range (start, stop):
        statement
        statement
        ...
        statement
```

header

body

- Set the variable equal to the start value
- Repeat the following:
    - Check if the **variable** is less than the stop. If not, stop.
    - Execute the **statements**.
    - Increase the variable's value by 1.

Control structures

- **Control structure**: a programming construct that affects the flow of a program's execution
- Controlled code may include one or more statements
- The `for` loop is an example of a looping control structure

Repetition over a range

```
print("1 squared = " + str(1 * 1))
print("2 squared = " + str(2 * 2))
print("3 squared = " + str(3 * 3))
print("4 squared = " + str(4 * 4))
print("5 squared = " + str(5 * 5))
print("6 squared = " + str(6 * 6))
```

- Intuition: "I want to print a line for each number from 1 to 6"

- The `for` loop does exactly that!

```
for i in range(1, 7):
    print(str(i) + " squared = " str(i * i))
```

- "For each integer **i** from 1 through 6, print ..."

Loop walkthrough

```
for i in range(1, 5):
    print(str(i) + " squared = " + str(i * i))

print("Whoo!")
```

Output:

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!
```

Multi-line loop body

```
print("+-----+")
for i in range(1, 4):
    print("\\    /")
    print("/    \\")
print("+-----+")
```

- Output:

```
+-----+
\     /
/     \
\     /
/     \
\     /
/     \
+-----+
```

Expressions for counter

```
high_temp = 5
for i in range(-3, high_temp // 2 + 1):
    print(i * 1.8 + 32)
```

- Output:

```
26.6
28.4
30.2
32.0
33.8
35.6
```

Rocket Exercise

- Write a method that produces the following output:

```
T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
blastoff!
The end.
```

print('',end='')

- Adding `,end=''` allows you to print without moving to the next line
  - allows you to print partial messages on the same line

```
high_temp = 5
for i in range(-3, high_temp // 2 + 1):
    print(i * 1.8 + 32, end=' ')
```

- Output:

```
26.6  28.4  30.2  32.0  33.8  35.6
```

  - Either concatenate `'   '` to separate the numbers or set `end='   '`

Changing step size

- Add a third number to the end of range, this is the step size
  - A negative number will count down instead of up

```
print("T-minus ")
for i in range(10, 0, -1):
     print(str(i) + ", ", end="")
print("blastoff!")
print("The end.")
```

  - Output:

```
T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
The end.
```

Exercise

- Write code to output these two figures using string multiplication and loops.

```
+/\/\/\/\/\/\/\/\/\/\/\+
|                      |
|                      |
|                      |
|                      |
|                      |
+/\/\/\/\/\/\/\/\/\/\/\+
```

```
+/\/\/\/\+
|        |
|        |
+/\/\/\/\+
```

Constants

- **constant**: A fixed value visible to the whole program.
  - value should only be set only at declaration; shouldn't be reassigned

- Syntax:
  - Just like declaring a normal variable:

    name = **value**

  - name is usually in ALL_UPPER_CASE

  - Examples:

    ```
    DAYS_IN_WEEK = 7
    INTEREST_RATE = 3.5
    SSN = 658234569
    ```

Constants and figures

- Consider the task of drawing the following scalable figure:

```
+/\/\/\/\/\/\/\/\/\/\/\+
|                      |
|                      |
|                      |
|                      |
|                      |
+/\/\/\/\/\/\/\/\/\/\/\+
```

Multiples of 5 occur many times

```
+/\/\/\/\+
|        |
|        |
+/\/\/\/\+
```

The same figure at size 2

Constant tables

```
SIZE = ...
```

- What equation would cause the code to print:

```
2 7 12 17 22
```

- To see patterns, make a table of `SIZE` and the numbers.
  - Each time `SIZE` goes up by 1, the number should go up by 5.
  - But `SIZE * 5` is too great by 3, so we subtract 3.

SIZE	number to print	5 * SIZE	5 * SIZE - 3
1	2	5	2
2	7	10	7
3	12	15	12
4	17	20	17
5	22	25	22

Constant tables question

- What equation would cause the code to print:

```
17 13 9 5 1
```

- Let's create the constant table together.
  - Each time `SIZE` goes up 1, the number printed should ...
  - But this multiple is off by a margin of ...

`SIZE`	number to print	`-4 * SIZE`	`-4 * SIZE+ 21`
1	17	-4	17
2	13	-8	13
3	9	-12	9
4	5	-16	5
5	1	-20	1