1 Basics

- **Quotes** — Use quotes when you want text and leave them out if you want to compute something. The first line below prints the text 2+2 and the second computes $2+2$ and prints 4.

```
print("2+2")
print(2+2)
```

- **Getting input** — Use `eval` with `input` when getting a number, `input` alone when getting text.

```
name = input("Enter your name: ")
num = eval(input("Enter a number: "))
```

- **Optional arguments to the print function**
  - `sep` — used to change or get rid of spaces that `print` inserts

```
print(1, 2, 3, 4, 5, sep='')   # prints 12345
print(1, 2, 3, 4, 5, sep='/')  # prints 1/2/3/4/5
```

  - `end` — used to change or get rid of when print jumps to next line

```
for i in range(10):
    print(i, end="")
```

    This prints `0123456789` all on the same line. Changing `end=""` to `end=" "` would put things on same line, separated by spaces.

- **Math formulas**
  - `+`, `-`, `*`, and `/` are the four basic math operators. Also:

Operator	What it does	Examples
`**`	Powers	`5**2` = 25
`%`	Modulo (remainder)	`19 % 5` = 4
`//`	Integer division	`5//3` = 1

  - A math formula like $3x + 5$ needs the times symbol, like this:

```
y = 3*x + 5
```

  - Order of operations matters. Use parentheses if necessary. An average calculation:

```
avg = (x + y + z) / 3
```

  - To use common math functions, import them from the `math` module. For instance,

```
from math import sin, pi
print("The sine of pi is", sin(pi))
```

  - The absolute value and round functions don't need to be imported:

```
print("The absolute value of -3 is", abs(-3))
print("4/3 rounded to two decimal places is", round(4/3, 2))
```

- **Random numbers**

```
from random import randint
x = randint(1, 5)
```

The import line goes near the top of your program. Use `randint(a,b)` to get a random number from `a` to `b` (including `a` and `b`).

2 If statements

- **Common if statements**

```
if x == 3:                          # if x is 3
if x != 3:                          # if x is not 3
if x >= 1 and x <= 5:               # if x is between 1 and 5
if x == 1 or x == 2:                # if x is 1 or 2
if name == "elvis":                 # checking strings
if (x == 3 or x == 5) and y == 2:   # use parens in complicated statements
```

- **if/else**

```
if guess == num:
    print("You got it!")
else:
    print("You missed it.")
```

- **if/elif/else** is useful if something can be one of several possibilities.

```
if entry == "yes":
    print("Your total bill is $50.00")
elif entry == "no":
    print("Your total bill is $45.00")
else:
    print("Invalid entry")
```

3 For Loops

- **Basics**
  - This will print `hello` 50 times:

```
for i in range(50):
    print("hello")
```

  - Indentation is used to tell what statements will be repeated. The statement below alternates printing `A` and `B` 50 times and then prints `Bye` once.

```
for i in range(50):
    print("A")
    print("B")
print("Bye")
```

  - The `i` variable in a for loop keeps track of where you are in the loop. In a simple for loop it starts off at 0 and goes to one less than whatever is in the range. The example below prints out the value of `i` at each step and will end up printing the numbers from 0 to 49.

```
for i in range(50):
    print(i)
```

- **The range statement** — Here are some example ranges:

Statement	Values generated
`range(10)`	`0,1,2,3,4,5,6,7,8,9`
`range(5,10)`	`5,6,7,8,9`
`range(1,10,2)`	`1,3,5,7,9`
`range(10,0,-1)`	`10,9,8,7,6,5,4,3,2,1`

One tricky thing is that the endpoint of the range is never included in the numbers generated. For instance, `range(1,10)` stops at 9.

4 Strings

- **String basics**

Statement	Description
`s = s + "!!"`	Add `!!` to the end of s
`s = "a"* 10`	Like s = `"aaaaaaaaaa`
`s.count(" ")`	Number of spaces in the string
`s = s.upper()`	Changes s to all caps
`s = s.replace("Hi","Hello")`	Replaces each `'Hi'` with `'Hello'`
`s.index("a")`	Location of the first `'a'` in s
`if "a" in s:`	See if s has any a's
`if s.isalpha():`	See if every character of s is a letter
`if s.isdigit():`	See if every character of s is a digit
`len(s)`	How long s is

- **String indices and slices** — Suppose s=`"abcdefghij"`:

Statement	Result	Description
`s[0]`	`a`	first character of s
`s[1]`	`b`	second character of s
`s[-1]`	`j`	last character of s
`s[-2]`	`i`	second-to-last character of s
`s[:3]`	`abc`	first three characters
`s[-3:]`	`hij`	last three characters
`s[2:5]`	`cde`	characters at indices 2, 3, 4
`s[5:]`	`fghij`	characters from index 5 to the end
`s[:]`	`abcdefghij`	entire string
`s[1:7:2]`	`bdf`	characters from index 1 to 6, by twos
`s[::-1]`	`jihgfedcba`	s in reverse

Warning: Slices are like `range` in that they don't include the last index.

- **Using slices and the index method together**

Statement	Description
`s[s.index("")+1]`	The character after the first space
`s[ : s.index("")]`	All the characters before the first space
`s[s.index("")+1 : ]`	All the characters after the first space

- **Converting numbers to strings**

  Use `str` to convert a number to a string. One common use for this is to access individual digits of a number:

```
num = 3141592654
s = str(num)
print("The first digit of num is", s[0])
```

  Another use for `str` is if we need to put a number into a string with some other stuff:

```
x = 3
s = "filename" + str(x) + ".txt"
```

  This might be useful in a program that needs to create a bunch of files with similar names.

- **Converting strings to numbers**

```
num = int(s)    # convert s to an integer
num = float(s)  # convert s to a decimal number
```

- **A few practical string examples**

Example	Description
`s[0].upper() + s[1:]`	capitalize first letter, leave rest unchanged
`s = s.replace("$","")`	remove all dollar signs
`int(s[-4:])`	convert last 4 characters to a number
`if s[-1].isalpha():`	if the last character is a letter

- **First way to loop through strings**

```
for i in range(len(s)):
    print(s[i])
```

The variable `i` keeps track of the location in the string; `s[i]` is the character at that location.

Example: Alternate printing the letters from two strings, `s` and `t`, of same length.

```
for i in range(len(s)):
    print(s[i], t[i], end = "")
```

- **Second way to loop through strings**

```
for c in s:
    print(c)
```

The loop variable `c` here takes on the values of each character in the string, one by one.

Example: Count how many letters there are in `s`.

```
count = 0
for c in s:
    if c.isalpha():
        count = count + 1
print("There are", count, "letters in s.")
```

Be careful not to mix the two types of looping. If you have a loop like `for i in s:` and then try to do something with `s[i]`, it will not work. The second type of loop is just a convenient shortcut for the first type in some situations.

- **Special characters**

Character	Effect
`'\n'`	Newline character, advances text to next line
`'\t'`	Tab character
`'\"'` and `'\''`	Useful if you need quote characters inside strings

- **String formatting**

String formatting is used to make things look nice for printing. Here are some examples:

Formatting code	Comments
`"{:.2f}".format(num)`	forces `num` to display to exactly 2 decimal places
`"{:10.2f}".format(num)`	`num` to 2 decimal places and lined up
`"{:10d}".format(num)`	use `d` for integers and `f` for floats
`"{:10s}".format(name)`	use `s` for strings

Note: in the above examples, the 10 stands for the maximum size of a number/string, but you would want to replace it with whatever the size is of the largest number/string you are working with. The effect of using the 10 (or whatever) is that Python will allocate 10 spots for displaying the number/string, and anything not filled up by the number will be filled by spaces, effectively left-justifying strings and right-justifying numbers

5 Lists

- **Things that work the same way for lists as for strings**
  - Indices and slices
  - The `in` operator for checking if something is in a list/string
  - The `count` and `index` methods
  - `+` and `*` are used for adding and repeating lists
  - Looping
- **Useful functions that work on lists**

Function	Result
`max(L)`	returns the largest item in `L`
`min(L)`	returns the largest item in `L`
`sum(L)`	returns the sum of the items in `L`
`len(L)`	returns the number of items in `L`

- **A few useful list methods**

  Assume the list is called `L`.

Method	Description
`L.append(x)`	adds `x` to the end of the list
`L.sort()`	sorts the list
`L.count(x)`	returns the number of times `x` occurs in the list
`L.index(x)`	returns the location of the first occurrence of `x`
`L.reverse()`	reverses the list
`L.remove(x)`	removes first occurrence of `x` from the list
`L.pop(p)`	removes the item at index `p` and returns its value
`L.insert(p,x)`	inserts `x` at index `p` of the list

- **To change a single element in a list**

  The following changes the first element in the list `L` to 99.

```
L[0] = 99
```

- **To build up a list one item at a time**

  Here is a program that builds up a list of 100 random numbers from 1 to 20:

```
from random import randint

L = []
for i in range(100):
    L.append(randint(1,20)
```

  The general technique for building up a list is to start with an empty list and append things in one at a time.

- **Using `input` to get a list**

  If you want to ask the user to enter a list, you can use something this:

```
L = eval(input("Enter a list"))
```

  The user needs to enter the list exactly as you would put it into a Python program, using brackets. If you replace `eval` with `list`, the user can omit the brackets.

- **Getting random things from lists**

  These functions are useful. To use them, you need to import them from the `random` module.

Statement	Result
`choice(L)`	returns a random item from `L`
`sample(L, n)`	returns `n` random items from `L`
`shuffle(L)`	puts the items of `L` in random order

- **The split method**

  The split method is a very useful method for breaking a string up. For example:

```
"s=abc de fgh i jklmn"
L = s.split()
print(L[0], L[-1])
```

  The above prints out `abc` and `jklmn`.

  By default, split will break things up at whitespace (spaces, tabs, and newlines). To break things up at something different, say at slashes, use something like the following:

```
s = "3/14/2013"
L = s.split("/")
day = int(L[0])
```

  The `int` in the last line is needed if we want to do math with the day.

6 Functions

Functions are useful for breaking long programs into manageable pieces. Also, if you find yourself doing the same thing over and over, you can put that code into a function and just have it in one place instead of several. This makes it easier to change things since you just have to change one place instead of several. Here is a simple function:

```
def draw_box():
    print("**********")
    print("**********")
    print("**********")
```

This would be put somewhere near the top of your program. You can add *parameters* to make the function customizable. To allow the caller to specify how wide the box should be, we can modify our function to the following:

```
def draw_box(width):
    print("*" * width)
    print("*" * width)
    print("*" * width)
```

Below is how we would call each function from the main part of the program:

```
draw_box()     # original function
draw_box(10)   # function with a parameter
```

Functions can also be used to do stuff and return a value. For instance, the following function gets a bill amount and tip amount from the caller and returns the total bill amount:

```
def compute_bill(bill, tip):
    total = bill + bill * tip/100
    return total
```

Here are two ways we could call the function:

```
print(compute_bill(24.99, 20))
x = compute_bill(24.99, 20) / 2
```

In the second way of calling the function, we see what the return does. It returns the value of the computation and allows us to do something with it (in this case dividing it by 2).

7 While loops

For loops are used for repeating actions when you know ahead of time how many times things have to be repeated. If you need to repeat things but the number of times can vary, then use a while loop. Usually with a while loop, you keep looping until something happens that makes you stop the loop.

Here is an example that keeps asking the user to guess something until they get it right:

```
guess = 0
while guess != 24:
    guess = eval(input("What is 6 times 4? "))
    if guess != 24:
        print("Wrong!  Try again.")
print("You got it.")
```

Here is an example that loops through a list `L` until a number greater than 5 is found.

```
i = 0
while L[i] <= 5:
    i = i + 1
```

When we loop through a list or string with a while loop, we have to take care of the indices ourselves (a for loop does that for us). We need our own variable `i` that we have to set to 0 before the loop and increment using `i = i + 1` inside the loop.

8 Working with text files

- **Reading from files** — The line below opens a text file and reads its lines into a list:

  ```
  L = [line.strip() for line in open('somefile.txt')]
  ```

  1. Suppose each line of the file is a separate word. The following will print all the words that end with the letter `q`:

     ```
     for word in L:
         if word[-1] == 'q':
             print(word)
     ```

  2. Suppose each line of the file contains a single integer. The following will add 10 to each integer and print the results:

     ```
     for line in L:
         print(int(line) + 10)
     ```

  3. Suppose there are multiple things on each line; maybe a typical line looks like this:

     ```
     Luigi, 3.45, 86
     ```

     The entries are a name, GPA, and number of credits, separated by commas. The line below prints the names of the students that have a GPA over 3 and at least 60 credits:

     e

     ```
     for line in L:
         M = line.split(",")
         if float(M[1]) > 3 and int(M[2]) >= 60:
             print(M[0])
     ```

- **Writing to text files** — Three things to do: open the file for writing, use the print statement with the optional `file` argument to write to the file, and close the file.

  ```
  my_file = open("filename.txt", "w")
  print("This will show up in the file but not onscreen", file=my_file)
  my_file.close()
  ```

  The `"w"` is used to indicate that we will be writing to the file. This code will overwrite whatever is in the file if it already exists. Use an `"a"` in place of `"w"` to append to the file.

9 Dictionaries

A dictionary is a way to tie two groups of data together. For example:

```
d = {"jan":31, "feb":28, "mar":31, "apr":30}
```

To get the number of days in January, use `d["jan"]`. A dictionary behaves like a list whose indices can be strings (or other things). The month names are this dictionary's *keys* and the numbers of days are its *values*. Here are some dictionary operations:

Statement	Description
`d[k]`	get the value associated with key `k`
`d[k] = 29`	change/create a value associated to key `k`
`if k in d:`	check if `k` is a key
`list(d)`	a list of the keys
`list(d.values())`	a list of the values
`[x[0] for x in d.items() if x[1]==31]`	a list of keys associated with a specific value

You can loop through dictionaries. The following prints out the keys and values.

```
for k in d:
    print(k, d[k])
```

Dictionaries are useful if you have two lists that you need to sync together. For instance, a useful dictionary for a quiz game might have keys that are the quiz questions and values that are the answers to the questions.

A useful dictionary for the game *Scrabble* might have keys that are letters and values that are the point values of the letters.

10 Two-dimensional lists

Here is how to create a 3×3 list of zeros:

```
L = [[0, 0, 0],
     [0, 0, 0],
     [0, 0, 0]]
```

We use two indices to access individual items. To get the entry in row `r`, column `c`, use `L[r][c]`.

When working with two dimensional lists, two for loops are often used, one for the rows and one for the columns. The following prints a 10×5 list:

```
for r in range(10):
    for c in range(5):
        print(L[r][c], end=" ")
    print()
```

Here is a quick way to create a large two-dimensional list (this one is 100×50):

```
L = [[0]*50 for i in range(100)]
```

11 Programming techniques

Counting A really common programming technique is to count how many times something happens. The following asks the user for 10 numbers and counts how many times the user enters a 7.

```
count = 0
for i in range(10):
    num = eval(input("Enter a number: "))
    if num == 7:
        count = count + 1
print(count)
```

When counting, you will almost always have a `count=0` statement to start the count at 0, and a `count=count+1` statement to add one to the count whenever something happens.

Summing A related technique to counting is to add up a bunch of things. Here is a program that adds up the numbers from 1 to 100:

```
total = 0
for i in range(1, 101):
    total = total + i
print("1+2+...+100 is", total)
```

Finding maxes and mins If you have a list `L` you want the max or min of, use `max(L)` or `min(L)`, but if you need to do something a bit more complicated, use something like the example below, which finds the longest word in a list of words:

```
longest_word = L[0]
for w in L:
    if len(w) > len(longest_word):  # change > to < for shortest word
        longest_word = w
```

Swapping To swap the values of `x` and `y`, use the following

```
saved_x = x
x = y
y = saved_x
```

Or use the following shortcut:

```
x, y = y, x
```

Using lists to shorten code A lot of times you'll find yourself writing essentially the same thing over and over, like the code below that gets the name of a month.

```
if month == 1:
    name = "January"
elif month == 2:
    name = "February"
# 10 more similar conditions follow...
```

We can shorten this considerably by using a list of month names:

```
names = ["January", "February", "March", "April", "May", "June",
         "July", "August", "September", "October", "November", "December"]
name = names[month-1]
```

If you find yourself copying and pasting the same code with just small changes, look for a way to use lists to simplify things.