lesson 1

basics

what we'll cover

- data types - numbers, strings, boolean
- operators - + - * / // ** (), order of operations
- print and input

Numbers

Whole numbers are called integers or are of type int.

1 2 3 4 5 13456

Numbers with a decimal component are called floating point numbers and are of type float.

1.0 2.123 3.14

Operators

Operators are symbols we use to perform arithmetic on numbers in Python. Some operators may also be used to manipulate strings or words or other objects. Here are a few examples of arithmetic operators. The examples below use ints. If using ints in operations you will usually receive ints as the result, but in Python 3 the / division operator will return a float.

```
>>> 4 + 3
7
```

```
>>> 4 * 3
12
```

The result of 10 divided by 3 gives you a float or floating point number.

```
>>> 10 / 3
3.3333333333333335
```

Operators

// floor division operator, the result has no decimal component. It will return an int.

```
>>> 10 // 3
3
```

** power operator

```
>>> 3**2
9
```

Operators

Placing a - minus sign in front of a number makes it negative. The minus sign by itself is called a negative unary operator because it only needs one number to perform a calculation. There's also a positive or + plus sign unary operator, but you'll likely never use it.

```
>>> -4
-4

>>> -(4-5)
1

>>> +4
4

>>> +(4-5)
-1
```

Order of Operations

You are probably familiar with order of operations. Python follows order of operations. For example: Here 3 * 5 is calculated before 2 is added to it.

```
>>> 2 + 3 * 5
17
```

2 to the 3rd power is calculated before 5 is multiplied to the result.

```
>>> 2**3 * 4
32
```

```
>>> 12.56 / 2**2
3.14
```

```
>>> 2 * 3 - 4 / 2
4.0
```

Order of Operations

You may use parentheses to group operations, overriding normal order.

```
>>> (2 + 4) * 3
18
```

```
>>> (4 * 2)**2
64
```

```
>>> (9  + 3) * (9 + 6)
180
```

Order of operations is:

```
+-(unary)  ** ()   *   /   //   +  -
```

Variables

You've seen a few variables in the previous exercises. Variables are labels or tags or names that you can assign to a number or to a math operation. They let you store a number or result to use later in your program.

```
>>> some_number = 4

>>> some_number
4

>>> result = some_number * 8

>>> result
32

>>> radius = 5

>>> pi = 3.14159

>>> area = pi * radius**2

>>> area
78.53975
```

Strings

Strings are a group of characters. A character could be a number, letter, or symbol like an operator or punctuation. Strings can also be made up of characters from other languages. Strings are identified either by ' ' single quotes or " " double quotes.

```
>>> "what do you call someone with no nose and no body"
'What do you call someone with no nose and no body?'
```

```
>>> 'nobody knows'
'nobody knows'
```

Strings may be stuck together to make bigger strings. This is called concatenation. Concatenation uses the + plus operator.

```
>>> 'Big' + ' ' + 'Ben'
'Big Ben'
```

Strings

Having the option of a single or double quote let's you easily add single or double quotes to your string.

```
>>> "Joe's car"
"Joe's car"
```

```
>>> 'John said, "Get off your horse and drink your milk!"'
```

Variables can also hold strings.

```
>>> first_name = 'Steve'
```

```
>>> last_name = 'Martin'
```

```
>>> full_name = first_name + ' ' + last_name
```

```
>>> full_name
'Steve Martin'
```

Strings

You may use the multiply operator to create a long string from a small one.

```
>>> short_name = "three "

>>> short_string = "three "
```

```
>>> long_string = short_string * 3

>>> long_string
'three three three '
```

You may use formatting to create a larger string.

```
>>> "four : %d" % 4
'four : 4'
```

Strings

You may pull individual letters from strings by adding [ ] square brackets to the end of the string and specifying the position of the character that you want to pull between the brackets. The first position is 0.

```
>>> name = "Jack"
```

```
>>> name[0]
'J'
```

```
>>> name[1]
'a'
```

```
>>> name[2]
'c'
```

```
>>> name[3]
'k'
```

Statements and Expressions

- An expression is anything that evaluates or can be translated into a value:
  - “Hello Fellow Nerds”
  - 8 + 5 // 2
  - 4 == 3
- A statement will act on an expression to do something
  - print(“I eat cheezbrger”)
  - name = “Steve”
  - if 2==2

Lists

A list in Python is grouping or collection of objects, such as a collection of strings. For example you could have a list that looks like this:

```
    pokemon = ["Bulbasaur", "Squirtle", "Charmander"]
```

You might also have a list of numbers, like this:

```
    ages = [10, 13, 20, 5]
```

You can pull individual items from a list using [ ] square brackets, passing in the index of the value you would like returned. As with strings this index begins at 0.

```
    ages[0]
    10

    pokemon[2]
    'Charmander'
```

Booleans

Think of a boolean as True or False, yes or no. Booleans can be identified using comparison operators.

```
   ==   !=   <   >   <=   >=
```

```
   >>> small = 3
```

```
   >>> small == 3
   True
```

```
   >>> small == 4
   False
```

```
   >>> large = 5
```

```
   >>> small == large
   False
```

```
   >>> small > large
   False
```

```
   >>> small < large
   True
```

```
   >>> small <= large
   True
```

```
   >>> small >= large
   False
```

```
   >>> small != large
   True
```

Boolean Expressions

You may assign a boolean value like True or False to a variable.

```
>>> b = True
```

```
>>> b == True
True
```

```
>>> b == False
False
```

There are some special boolean operations in python.

```
>>> 'J' in 'Jack'
True
```

```
>>> 'J' not in 'Jack'
False
```

```
>>> 'k' not in 'Jack'
False
```

Boolean Expressions

The and operator will return true if the comparisons on both sides are True. The or operator will return true if one of the two comparisons surrounding it is True.

```
>>> x = 3
>>> y = 4

>>> x == 3 and y == 5
False

>>> x == 3 or y == 5
True
```

Print

Use print to write strings to the screen. You may print messages and string variables. You may also print other data, like numbers and booleans, if you convert those to strings before printing. To print, surround the text you would like to print with ( ) parentheses.

```
>>> print("hello")
hello
```

```
>>> print("hello " + "John")
hello John
```

Numbers may be printed by themselves.

```
>>> print(4)
4
```

Print

Strings and numbers cannot be added together.

```
>>> print("num: " + 4)
Traceback (most recent call last):
  File "stdin", line 1, in
TypeError: Can't convert 'int' object to str implicitly
```

You may print strings and ints together using a , comma to separate the values.

```
>>> print("num: ", 4)
num: 4
```

It's more common to format a string that you plan to print.

```
>>> print("num: %d" % 4)
num: 4
```

Input

You may ask for a value and assign it to a variable using the input function.

```
>>> name = input('What is your name? ')
What is your name? John

>>> print("Hello %s" % name)
Hello John
```

Input

The input function returns a string. If you would like to use the string returned as a number pass it to the int() function.

```
>>> age = input("How old are you? ")
How old are you? 8

>>> age
'8'

>>> age * 3
'888'

>>> int(age) * 3
24
```

Review 1

Integers

2 3

Floats

2.0 3.14

Strings

“Jack’s Car”

‘Jack\’s Car’

Variables

value = 3

name = “Sarah”

Operators

+ - * / // ** % ()

(3 + 4) * (2 + 9) = 18

3 + 4 * 2 + 9 = 20

Booleans

True or False

== != < > <= >= in

4 < 5 True, 4 < 3 False

“Ball” != “Bike” True

“J” in “Jack” True

num=3 num < 4 True

(num < 4) and (name == “Sarah”)

(num > 10) or (name == “Sarah”)

Lists

colors = [“red”, “green”, “blue”]

colors[0]

colors.append(“yellow”)

lengths = [123, 30, 150, 142, 200]

lengths[3]

Print

print(“hello”)

print(“hello”, name)

print(“third”, length[3])

print(“hello %s” % name)

print(“third %d” % length[3])

Input

name = input(“What is your name? “)

print(“hello”, name)

answer = input(“What is your age? “)

age = int(answer) + 1

print (“your age next year is”, age)

today

- dictionaries
- code blocks
- control statements - if, while, for
- functions
- exercise

Dictionaries

A Python dictionary is data structure that lets the programmer map a key or term to a value or definition.

```
animals = {
    'cow': 'four legged bovine, vegan, sometimes spotted',
    'cat': 'four legged feline, omnivore, usually furry, meows alot',
    'bird': 'winged creature, noisy, tweets constantly'
}
```

```
>>> print(animals)
{'cat': 'four legged feline, omnivore, usually furry, meows alot', 'cow': 'four legged
bovine, vegan, sometimes spotted', 'bird': 'winged creature, noisy, tweets constantly'}
```

To find the definition or value of a particular term or key, you may reference the key using the square brackets. This looks similar to the way we referenced items in a list or letters in a string.

```
>>> animals['cow']

four legged bovine, vegan, sometimes spotted
```

More Dictionaries

We can create dictionaries as we did above with animals. We might also want to create an empty dictionary and add values to it.

```
>>> animals = {}

>>> animals['dog'] = 'not a cat, a cow, or a bird, barks a lot'

>>> animals['dog']
'not a cat, a cow, or a bird, barks a lot'
```

Code Blocks

Code blocks are statements grouped together, executed in sequence, usually to perform a single task. Code blocks can be nested within one another using control statements like if, while or for or in function and class definitions.

Python is an indented language. What this means is that sections of code that we run begin and end with an indentation. Python code blocks are typically indented with 4 spaces.

```
def write_client(filename, functions):
""" Create a client module from a list of function objects. """
module_name = os.path.splitext(os.path.split(filename)[1])[0]
with open(module_name + '_client' + '.py', "wt") as oout_file:
    oout_file.write(client_header % end_point)
    for function in functions:
        arguments = ''
        for i, arg in enumerate(function['arguments']):
            arguments += arg if i == 0 else ', ' + arg
        oout_file.write("def %s(%s):\n" % (function['name'], arguments))
        oout_file.write("    return g('%s', %s)['return_value']\n\n\n" % (function['name'], arguments))
    oout_file.write(client_footer)
```

if statement

Use the if statement and a comparison to decide what code blocks should run.

```
if True:
    some_number = 2
    print('this code ran', some_number)

if False:
    another_number = 4
    print('did this code run?', another_number)

some_number = 2
if some_number <= 2:
    another_number = some_number * 4
    print('this code ran', another_number)
```

if else and elif

Use the keyword else to identify a condition other than the one identified in your if comparison.

```
some_number = 3
if some_number > 4:
    print('some_number is greater than 4')
else:
    print('some_number is less than or equal to 4')
```

The elif (else if) keyword will let you chain together comparisons. Again this makes your code easier to understand and ensures that only one of the code blocks runs.

```
some_number = 3
if some_number > 4:
    print('some_number is greater than 4')
elif 2 > some_number <= 4:
    print('some_number is greater than 2 and less than or equal to 4')
elif some_number > 0:
    print('some_number is greater than 0')
else:
    print('some_number is less than or equal to 0')
```

while loop

If you need repeatedly run a code block use a loop. Like the if statement the while loop uses a comparison. If the comparison is True the code block will continue to run.

```
some_number = 0

while some_number < 100:
    some_number = some_number + 10
    print('some_number is less than 100', some_number)

print('some_number is now', some_number)
```

for loop

Another type of loop is the for loop. The for loop does not use a comparison like the while loop. The for loop is useful when you would like to use a counter in your code block. Also useful if you would like to pull individual values from a list or collection into your code block. This is called iteration.

A common use of for is to execute a block of code a certain number of times using the range() function.

```
sum = 0
for number in range(10):
    sum = sum + number
    print('number',number,'sum',sum)

number 0 sum 0
number 1 sum 1
number 2 sum 3
number 3 sum 6
number 4 sum 10
number 5 sum 15
number 6 sum 21
number 7 sum 28
number 8 sum 36
number 9 sum 45
```

for loop, lists and dictionaries

For loops can walk through or iterate through a list in order.

```
names = ['Steve', 'Ned', 'Klaus', 'Jane']
for name in names:
    if name == 'Klaus':
        print('Bye',name)
    else:
        print('Hi',name)

Hi Steve
Hi Ned
Bye Klaus
Hi Jane
```

We may also iterate through dictionaries. Unlike lists, dictionaries are not sorted in any particular order.

```
animals = {
    'cow': 'four legged bovine, vegan, sometimes spotted',
    'cat': 'four legged feline, omnivore, usually furry, meows alot',
    'bird': 'winged creature, noisy, tweets constantly'
}

for k, v in animals.items():
    print(k,' - ',v)
    if k == 'cat':
        print('meow')

cat  -  four legged feline, omnivore, usually furry, meows alot
meow
bird  -  winged creature, noisy, tweets constantly
cow  -  four legged bovine, vegan, sometimes spotted
```

Functions

Functions are sections of code that we can reuse throughout our program. Functions usually are defined to do one thing really well. A few of the functions we've used are the print(), input(0 and range() system functions.

You may define your own functions. The first line of a function is where we name the function. The def keyword is used to identify the name of a function. It comes before the name. Parentheses come after the name of the function. The code that makes up a function is in an indented block below this line. For example:

```
    def say_hello():
        print('Hello')
```

To run or call a function just type the name and the parentheses.

```
    >>> def hello():
    ...     name = input('Name? ')
    ...     print('Hello', name)
    ...
    >>> hello()
    Name? Jack
    Hello Jack
    >>>
```

Function Parameters

Parameters are values that you pass into a function from outside a function. You've done this previously with the print(), input() and range() functions. To add parameters to your user defined function add variable names within the parentheses that come after the function name. These variables will be available within the code block of your function.

```
>>> def hello(name):
...     print('Hello',name)
...
>>> hello('John')
Hello John
```

```
>>> def to_third_power(num):
...     print(num**3)
...
>>> to_third_power(2)
8
```

You may define functions that allow multiple parameters.

```
>>> def power(num, pow):
...     print(num**pow)
...
>>> power(2, 3)
8
```

You may reference the parameter name when calling the function.

```
>>> def power(num, pow):
...     print(num**pow)
...
>>> power(pow=2, num=3)
```

Functions returning results

Functions may return results. These results can be displayed on the screen or used in other calculations.

```
>>> def power(num, pow):
...     return num**pow
...
>>> power(2, 3)
8
>>> 4 + power(2,3)
12
>>> print('power 2**3 =', power(2, 3))
power 2**3 = 8
```

In Python functions may return multiple values.

```
>>> def hello(name, age):
...     greeting = 'Hello' + name
...     in_ten_years = age + 10
...     return greeting, in_ten_years
>>> g,t = hello('John', 10)
>>> print(g, t)
Hello John 20
```

Review 2

dictionary

```
nums = {‘one’:1, ‘two’: 2}
nums[‘two’]    2
nums[‘three’] = 3
```

code block

```
def f():
    if True:
        print(‘hi’)
```

if, else, elif

```
num = 3
if num == 4:
    print(‘is 4’)
elif num == 3:
    print(‘is 3’)
else:
    print(‘ugh’)
```

while loop

```
num = 0
while num < 3:
    num += 1
    print(num)
```

for loop

```
for i in range(3):
    print(i)

ls = [3,2,5]
for n in ls:
    print n

nums = {‘one’:1, ‘two’: 2}
for k,v in nums.items():
    print(k,’ - ‘,v)
```

function

```
def say_hi():
    print(‘hi’)
```

parameters

```
def say_hi(name):
    print(‘hi’, name)

def say_age(name, age):
    print(name,’is’,age)
say_age(‘Steve’, 3)
say_age(age=3,name=‘Steve’)
```

return

```
def add_two_numbers(n1, n2):
    sum = n1 + n2
    return sum
```

exercise - guessing game

1. Create a new python file named guess.py
2. On the first line import random module
3. Define function get_answer(), it should return the value random.randint(1,100)
4. Outside of function define two variables
    1. variable answer should equal result from get_answer()
    2. variable guess should be initialized to -1
5. Create a while statement that continues to run if guess and answer are not equal
6. Inside the while loop block do the following
    1. Using input() ask the user for their guess, “guess a number between 1 and 100”, store value in variable guess
    2. Convert guess to integer, int(), from string
    3. Using if, elif, and else tell the user if their number is too high, too low or correct.

Solution

```
import random


def get_answer():
    value = random.randint(1, 100)
    return value


answer = get_answer()
guess = -1

while guess != answer:
    guess = input("guess a number between 1 and 100: ")
    guess = int(guess)
    if guess > answer:
        print("Too high")
    elif guess < answer:
        print("Too low")
    else:
        print("Your answer",guess,"is correct!")
```