Remember

* In Python, division of two numbers – either floating-point numbers or integers – always yields a float for example 2/3 = 0.6666666666666 1/1 = 1.0
* Multiplication and division have higher precedence than addition and subtraction; operators are otherwise applied left to right. You can override this by adding parentheses.
* For built-in function – dir(\_\_builtins\_\_)
* For help(abs) – to functionality of any method or like help(list)
* The expression not not x is equivalent to x.
* If a function ends without a return statement being executed, that function returns None. Value None has type NoneType.

Operator Symbol

* Exponentiation \*\* e.g. 2\*\*5 = 32
* Integer Division // e.g. 2//3 = 0 (i.e. return integer part)

Operator Precedence

\*\* highest precedence

-(negation)

\* / // %

+ - lowest precedence

Syntax

* The rules that describe valid combinations of Python symbols
* A syntax error occurs when we an instruction with invalid syntax is executed.
* Syntax Error: 4 + 5) \*3 Invalid Syntax

Semantics

* The meaning of a combination of python symbols

Semantics Error:

* A semantic error occurs when an instruction with invalid semantics is executed.
* 1/0 zeroDivisionError: division by zero

Variable

* A named location in computer memory

Terminology

* A value has a memory address.
* A variable contains a memory address.
* A variable refers to a value.
* A variable points to a value.

Example of use

* Value 8.5 has memory address x34
* Variable shoe\_size contains memory address x34
* The value of shoe\_size is 8.5
* shoe\_size refers to value 8.5
* shoe\_size points to value 8.5

Assignment Statement

Variable = expression

Rules for executing an assignment statement

1. Evaluate the expression on the right of the = sign to produce a value. This value has a memory address.
2. Store the memory address of the value in the variable on the left of the =

Python naming convention

* Use pathole\_case in most situations so that other python programmers have an easier time reading your code.

Argument

* An argument is a value given to a function. To determine how many arguments should be given, look at the number of parameters in the function header.

Function

* def a keyword indicating a function definition
* Parameter: a variable that appears between the parentheses of a function definition. Parameters get their values from expressions in a function call.
* Return Statement

return expression

* Rules for executing a return statement

1. Evaluate the expression, which produces a value.
2. Produce that value as the result of the function call.

* Function definition

def functionname(parameters)

body

* Rules for executing an assignment statement

1. Evaluate the expression on the right of the = sign to produce a value.
2. Store the memory address of the value in the variable on the left of the =.

* Function Call

function\_name(arguments)

* Rules for executing a function call

1. Evaluate the arguments to produce memory addresses.
2. Store those memory addresses in the corresponding parameters.
3. Execute the body of the function.

Expression Description

* str1 + str2 concatenate str1 and str2 e.g. ‘ha’ + ‘bwa’ i.e. habwa
* str1 \* int1 concatenate int1 copies of str1 e.g. ‘ha’ \* 5 i.e. ‘hahahahaha’
* int1 \* str1 concatenate int1 copies of str1 e.g. 5 \* ‘ha’ i.e. ‘hahahahaha’
* The \* and + operators obey the standard precedence rules when used with strings.
* All other mathematical operators and operators result in a TypeError.
* Example : ‘bwa’ + ‘ha’\*5 i.e. bwahahahahaha
* For concatenating must be two are strings

Input/output and str Formatting

* print: print a sequence of arguments for user to read.
* Multiple arguments are separated by spaces like print(“hello”, “there”)
* During execution of a function call, if the end of the function body is reached without executing a return statement, that function call produces value none.
* input : get a string from the user.

The argument to input is called a prompt.

The function pauses until the user types a newline.

The return value is the string the user typed.

* age = input("How old are you? ") – Regardless of the text entered by the user, the input function returns str
* Triple-quoted strings can span multiple lines. print(’’’how

are

you?’’’)

* The print function puts spaces between its arguments when it prints them. Remember that quotation marks are how we tell Python that we're talking about a string, but they are not a part of the string (unless we put quotation marks somewhere in the string between the enclosing quotation marks).
* Since \ in a string is used to signify that the following character is to be interpreted in a special way, we need a way to represent the \ character itself.
* Remember that \n is used to specify that the newline character is to be included in the string. \n is only a representation of the character and is not the character itself.

Escape Sequences

* \n newline (ASCII linefeed – LF)
* \t tab (ASCII horizontal tab – TAB)
* \\ backslash (\)
* \’ Single quote (’)
* \’’ double quote(’’)

DocString

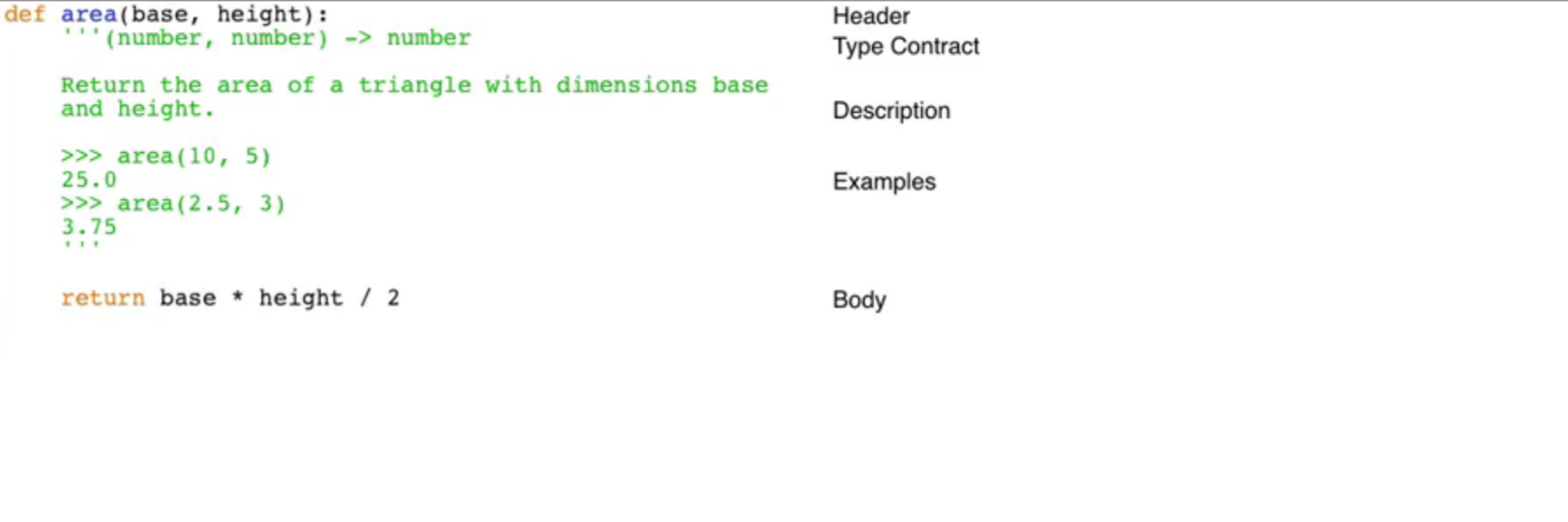
* Built-in function help displays the docstring from a function definition.
* For our own use this code snippet

'''(number, number) -> number

Return the area of a triangle with dimensions’ base and height.

'''

Design Recipe



* The Problem: The United States measures temperature in Fahrenheit and Canada measures it in Celsius. When travelling between the two countries it helps to have a conversion function. We’ll write a function that converts from Fahrenheit to Celsius.

Recipe for Designing Functions:

1. Examples

* What should your function do?
* Type a couple of example calls.
* Pick a name (often a verb or verb phrase):
* What is a short answer to “What does your function do?”

>>> convert\_to\_celsius(32)

0

>>>convert\_to\_celsius(212)

100

1. Type Contract

* What are the parameter types?
* What type of value is returned?

(number) -> float

1. Header

* Pick meaningful parameter names.

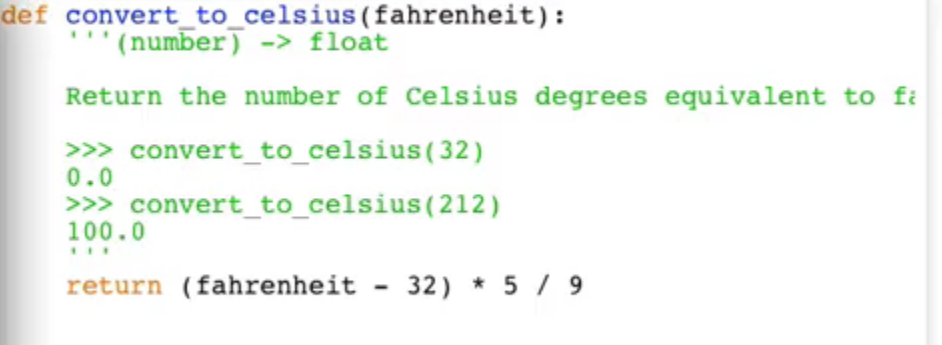
def convert\_to\_celsius(fahrenheit):

1. Description

* Mention every parameter in your description.
* Describe the return value.

1. Body

* return (fahrenheit – 32) \* 5 / 9



Recipe for Designing Functions:

* Examples
* Type Contract
* Header
* Description
* Body
* Test

Task: write a function to calculate the semiperimeter of a triangle. (That’s half of the perimeter.)

Stack Frame

* A region of computer memory for keeping track of information about a function being executed.
* When a function returns, its stack frame is removed from the call stack.
* When a function exits, its stack frame gets erased from memory. All local variables for that function, such as result, disappear.
* Variable result is assigned a value on line 8. On the right-hand side of the assignment statement on line 8, the value that num\_hours refers to is used. Num\_hours is the parameter to function conver\_to\_minutes and its value is set when the function is called. It's the function call on line 12 that is in the process of being executed during the visualization above. Check to see what the value is passed as an argument to that function call in order to determine the value of result.
* In an assignment statement, the variable on the left is created as the last step, so hours\_asleep won't exist until after line 11 has been fully executed.

Assignment statement and computer memory

Variable = expression

* If a variable does not exist in the current stack frame, Python creates it.

When a function exits

* The frame for the function is erased, and control passes back to the statement containing the function call. The value of that function call is whatever was returned by the function.

Logical operator. Operands are Boolean expressions

Logical Operator Symbol

not: not

and: and

or: or

Operator Precedence

not – highest precedence

and

or – lowest precendence

Type Conversion Functions

* str : return the information in the argument wrapped as a str. e.g. str(3)

🡪‘3’

* int : return the information in the argument wrapped as an int. e.g. int(‘456’) 🡪 456
* float : return the information in the argument wrapped as a float. e.g. float(‘456’) 🡪 456.0

Module

* a file containing function definitions and other statements
* A python file defines a module:*triangle.py* defines module *triangle*.
* Module *math* is part of Python, and contains functions that are math-related.
* When importing, use the module name. The filename includes the *.py* extension, but the module name does not.

import\_statement:

import module\_name e.g. import math for directory dir(math)

To access a function within a module:

module\_name.function\_name

if condition

* The **if** condition already evaluates to **True** or **False** so you can just return that Boolean expression.
* There are 2 situations in which the function returns **True**: when the temperature is above 28 and when the temperature is below 12. If either of those occur, the result is **True**. If neither occurs, the result is **False**.

Operator Symbol

String Comparison

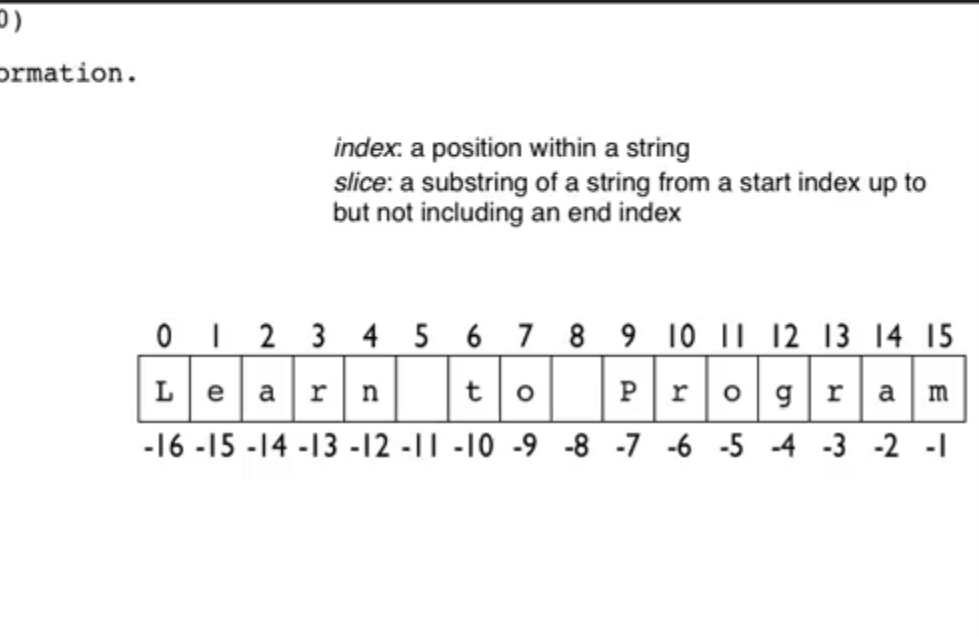
* Capital letters are not equal to their lowercase counterparts.
* Values of different types can be compared for equality.
* Values of different types usually cannot be compared for ordering. (Although **ints** and **floats** can be compared to each other.)
* len(str) : return the number of characters in **str**.

String

* index : a position within a string
* slice : a substring of a string from a start index up to but not including an end index.
* Positive indices count from the left-hand side with the first character at index 0, the second at index 1, and so on.
* Negative indices count from the right-hand side with the last character at index -1, the second last at index -2, and so on.
* The general form of a slice is:**s[start:end]** , where **start** and **end** refer to **int**s representing indices. The substring begins with the character at index **start** and goes up to but not included the character at index **end.**

Positive indices count from the left-hand side with the first character at index 0, the second at index 1, and so on.

Negative indices count from the right-hand side with the last character at index -1, the second last at index -2, and so on.



* Strings are immutable. They cannot be changed.
* Instead of modifying a string we need to assign new string with the previous string

Method

* A function inside of an object.

Form of a method call

object.method(arguments)

For String Method

* dir(str)
* help(str.lower)

for loop over a string:

For *variable* in *str*:

Body

For loop execution over empty string

* The number of times the body of the loop executes is equal to the length of the string. Because **len(“”)** is 0, a **for** loop over the empty string will execute the body 0 times. The empty string contains no characters.

**While loop:**

while expression:

Statements

iteration: execution of the loop body

For and While Loop

* The key difference between these loops is that the for loop checks every character in the string, but the while loop terminates as soon as it finds a vowel. So, the loops differ on any string where a consonant follows a vowel.

**List**

* General form of a list: [exp1, exp2, …, exp3]
* Lists can contain items of any type.
* Lists can be indexed and sliced.
* len(list): return the number of items in **list**.
* min(list): return the smallest item in **list**.
* max(list): return the largest item in **list**.
* **sum**(list): return the sum of items in **list** (items must be numeric).
* Lists may have items of different types.
* numbers = [[1,2], 3, [4,5]] len(numbers) will return 3
* In nested lists each sublist is treated as a *single* element regardless of its own length. *Numbers* therefore has three elements: one *int* and two *lists*.

**For loop over a list:**

for variable in list:

body

**List Methods**

* dir(list)

**List methods that modify the list**

* list.append(object)

append object to the end of the list

* list.extend(list)

append the items in the list parameter to the list

* list.pop()

remove and return the item at the end of the list (optional index to remove from anywhere)

* list.remove(object)

remove the first occurrence of the object; error if not there

* list.reverse()

reverse the list

* list.sort()

sort the list from smallest to largest

* list.insert(int, object)

insert object at the given index, moving items to make room

**List methods that do not modify the list**

* list.count(object)

return the number of times object occurs in list

* list.index(object)

return the index of the first occurrence of object; error if not there

**Function**

* A function or method has a side effect if it returns a value and also modifies an object.
* Any functions that modify parameters must explicitly say so in the docstring. If a docstring does not say that parameters are modified, then the function should not modify them.
* Any attempt to modify an immutable type will raise a **TypeError**

**Aliasing**

* When two or more variables refer to the same object.

**Mutable Types Immutable Types**

list int, float, str, bool

**Mutability**

* We say that lists are mutable: they can be modified. All the other types we have seen so far (str, int, float and bool) are immutable: they cannot be modified.

**Range**

* Range is used to generate a sequence of numbers that represent the indices of a string or list.
* range([start,] stop[, stop]) -> range object return a virtual sequence of numbers from start to stop by step.
* **range** takes (up to) 3 arguments: the start value (inclusive), the stop value (exclusive), and the step. This question needs to start at 2, stop when a value greater than 10 is reached (either 11 or 12), and step by 2 because we only want the even numbers. (0 is not a positive number and so should not be included.)
* **range**([start, ] stop[, step]) -> range object return a virtual sequence of numbers from start to stop by step.

File

* To open a file: open (filename, mode)
* Mode: ‘r’ to open for reading

‘w’ to open for writing (erasing what is already in the file)

‘a’ to open for appending (adding to what is already in the file)

* Method readline()
* Read and return the next line from the file, including the newline character (if it exists).
* Return the empty string if there are no more lines in the file.
* The readline approach
* Use readline to skip the beginning of the file
* Read the first line of the file you are interested in
* While not at the end of the interesting section:

Process the current line

Read the next line of the interesting section

When you want to process only some parts of a file.

* The for line in file approach

for line in file:

process line

When to use this approach

* When you want to process every line in a file.
* The read approach

file.read()

When to use this approach

When you want to read the whole file at once.  
- By calling **readline** once at the beginning of our code, we can easily discard the header.

* Because our file is 500,000 rows long, it's probably a bit too big to use **read** or **readlines**, since these involve reading the whole file into memory at once.
* **readlines** will give us a list of student names. If we call the **list.sort** method (which we saw in week 5) on this list, we'll get our list of student names sorted alphabetically.
* **Module tkinter**

Contains types and functions for creating graphical user interfaces

* **Module tkinter.filedialog**

Contains functions for creating file dialogs

**Write**

* After writing in file it will return how many characters written in the file
* The **write** method doesn't append any newlines automatically, so if we want breaks between the "lines" we're writing, we need to mention them explicitly.

**Tuple**

* General form of a tuple
* (expr1, expr2, …., expr3)
* Tuples can be indexed and sliced
* Tuples are *immutable* they can’t be changed.

**For loop over a tuple:**

For *variable* in *tuple*:

body

Len(list): return the number of items in list.

* **index**, **len**, and indexing all work on **tuple**s just as on **list**s. But **tuple**s have no **pop** method, because this is meant to return and remove the last element, and we are not allowed to modify a **tuple**.

**Dictionary**

* **dict:** Python’s dictionary type
* **G**eneral form of a dictionary: {key1 : value1, key2 : value2, …, keyN : valueN}
* The keys are the items on the left-hand side of each : and the values are on the right-hand side.
* Keys are used as indices to access their associated values.

**Operator Symbol**

in contains

del remove

* *len(dict):* return the number of key-value pairs in *dict*
* Dictionaries are mutable.
* Dictionaries are unordered.
* Keys must be immutable.
* Lists is not actable but tuple is acceptable
* When we iterate over a **dict**, we get its keys. So, **k** refers to a key, and to modify its associated value must first retrieve the old value using **d[k]** and assign a new value to **d[k]**.
* Operator **in**, when used with a dictionary, checks whether an object is a key in the dictionary. It doesn't look at the values. Also, **len** only counts keys.
* As we saw in the first lecture on **dict,** the keys to a **dict**, must be immutable (for example, we can’t use a **list**, as a key). However, values can be mutable. So, in order to invert a dictionary, it must have no mutable values.