1/15

Function definition: defines what a function will do

* Def Celsius\_to\_farenhietf(degrees : int) -> float:
* “”” convert degrees Celsius to degrees Fahrenheit.”””
* Return (degrees \* 9/5) + 32.

Function call expressions: use the definitions and executes the code in it to get an answer

* Celsius\_to\_farenhiet(degrees=0).

Fundamental Pattern of Functions:

* Frist step is use a function call expression

**A parameterized function definition:** is one where you can substitute input arguments, follow the same steps, and get different but intentional results

* Such as turning degrees Fahrenheit to Celsius

First thing you need to make a function

def name of function(parameter: type) -> returnType:

“””Docstring description of the function”””. – just makes the function more readable-

Return expression\_of\_type\_returnType

Function signature: Parameters: list of ingredients

Function body: second 2 lines

* Specifies the subprogram, or set of steps, which will be carried out every time a function calls the definitions
* Each statement must be indented by at least one level

Return statements: are special and written inside of the function definition

Anatomy of a function call

Name\_of\_function(argument=value)

1/17

The return statement vs. calls to print

The return statement is for your computer to send a result back to the function calls “bookmark” within your progress

The print statement shows the data the computer gets when you call on the function.

* Printing its return value directly with *print(my\_func())*  or
* By storing the returned value is a variable and later printing the variable

Tracing programs by hand: into memory diagrams

* Help us keep track of where we currently are in the program

This includes:

* The current line of code or expression in a line
* The trail of the function calls that led to the current line and frame of execution
* The names of the parameters/variables and a map of the values they are bound to.

Memory diagrams:

* A programs runtime environment is the mapping of names in your program to their locations in memory
* A programs state is made up of the values stored in those locations
* You can use memory diagrams to visually keep track of both the environment and its state.
* Memory diagrams will help you keep track of how funtioncalls are processed
  + Where the function was called
  + What was the return value and where was it returned
  + And more??\\!

1/27/2025

Warm-up questions:

10.0 == 10 is true even though one is a float and one an int

Relational operator practice:

1+2<3+4 : 3 < 7 : True

* Do the math first before dealing with relational operators

110.0 != 110 ; false

Every character in Python has an (ASCII) value associated with it

* UNC > DUKE : true because of ASCII values

**Or operator**

As long as one value is true then the whole expression is true

Expression:

* False or False: False
* True or False: True
* False or True: True
* True or True: True

**And operator**

Both expressions to the left and the right need to be true in order for the expression to be true

* True and True: the only true statement

**Not operator**

Negate the true or false value that is after it

* Not false: true
* Not true: false

Precedence High to low

0: arithmetic operators (pemdas)

1: relational operations

2. not

3. and

4. or

1/29/25

Control flow is linear : from the first line to the second to the third etc…

If-then/conditional statements::

If <condition>:

<then execute these statements

<rest of program>

A diagram of a decision

Description automatically generatedCannot enter both the else block and the then block, it is one or the other. Either way the program will keep moving on.

When executed the lines containing the function def, if, if return, and next statement will be executed

The else statement: function def, if statement, else statement, else return, and next statement will be executed.

General syntax and semantics:

1. When evalutation reaches an if statement, the Boolean test expression is evaluated.
2. If the expression evaluates to True, control continues into the *then statement block*. If the then statement block completes without a return, control continues by moving on to the next statement after the *if* statement.
3. Otherwise, if the test expression evaluates to Flase, the control jumps over the then block and continues to the next line, whether it is an else statement block or the next statement in the program.

1/31/25

f-strings (formatted string literals)

* Helpful way to embed expressions directly into strings

without f strings:

* Print(“they are “ + str(30 + 1))

With f-strings

* Print(f”they are {30 + 1}

Both will output: they are 31

F strings relieve us of the duty of having to use string concatenation

2/3/2025 elif and more conditionals practice

Elif (else if) statement lest us check multiple conditions sequentially without nested if-else statements

A screen shot of a computer code

Description automatically generated

2/5/25 Recursion and Positional Arguments

Signature vs call :

* Signature in the function
* The call calls back to the function. Also contains keyword arguments since you are assigning values based on the parameter names.

Positional arguments: values are assigned based on the order (position) of the arguments. IN your function call

A screen shot of a computer

Description automatically generated

Stack overflow and Recursion Error:

* When a programmer writes a function that calls itself indefinitely (infinitely), the function call stack will overflow
* This leads to a stack overflow or recursion error

Base Case and Recursive Case

* The key to writing recursive functions that are non-infinite
* To avoid stackoverflow errors
  + You must have at least one base case: a branch in a recursively defined function that does not recur

2/7/25.

Two new concepts:

Named constants and Default parameters:

Named constants:

* Variables that are meant to hold a value that doesn’t change throughout the programs execution
* Naming convention: all uppercase letters with underscores between any words

Default Parameters:

* A parameter in a function signature that is assigned a defalt value. If a function call does not provide a value for that parameter, the defalt value is used.
* Default parameters should come after non-default parameters in the function signature.

2/17/2025 \

Variable assignment: associates a name with a data type and a space in the current frame

* Binds a new value to a variable name in memory
* Declaration : <name>: <type> (associates a name/identifier with a data type, and a space in the current frame)

Variable assignment: students = 300 (binds a new value to a variable name in memory )

Initialization: students: int = 300 (for the first time you assign that variable

Variable access: “reading” or using a variable name in an expression

Left-hand vs right-hand side of assignment:

* Each side of the assignment operator (=) plays a distinct role in a variable assignment.

2/21/25

While loop statements

A diagram of a test condition

Description automatically generatedWhen using a while loop statement in code, when the condition evaluates to true, the repeat block is repeated and it jumps back up to the test condition to see if the condition is still true,

When the condition evaluates to false, skip past the repeat block and continue on to the next line of code at the same level of indentation as the while keyword.

It can be made where the while loop never becomes false and continues infinitely. So it must be made sure that the loop will eventually be false.

Common uses of the while loop

* User imput validation: prompt the user for a valid input until they give one to you
  + Like the wordle game code
* Game loops: keep a game running until some condition is met
  + Such as if you run out of lives
* Iterating through values
  + Counting from 0 to n
  + Looping through every character in a string (via subscription notation)
* \

2/24

Relative reassignment operators

* It is very common to need to update the value of a variable, relative to its current calue

Count: int = 1

Count = count + 1

Relative reassignment operators offer a shorthand way of doing this

* Count += 1. (increases the value by 1)
* Count -= 1 (decreases the value by one)

2/26 **Lists**

A list is a data structure something that lets you organize and soter data in a format such that they can be accessed and precessed efficiently Lists are mutable meaning their values can be changed after intializationA black rectangular box with white text

Description automatically generated

Declaring a type of list:

With a constructor:

* <list name>: list[<item type>] = list()
* Grocery\_list: list[str] = list()

With a literal:

* <listname>: list[<item type>] = []
* Grocery\_list: list[str] = []

Adding an item to the end of a list:

* <list name> .append(<item>)
* Grocery\_list.append(“bannanas”)

Indexing

* Grocery\_list: str[str] = [“bannanas’, “milk”, “bread”]
* Grocery\_list[0]
* Starts with a 0 like strings

Modifying by index

* Grocery\_list: list[str] = [“bannas”, “milk”, “bread”]
* Grovery\_list[1] = “eggs”
* The “milk” would be updated to say “eggs”
* Len(grocery\_list)=. 3 because there are 3 items

Remove an item from a ist – “pop off!”

* Grocery \_list: list[str] = [“eggs”, “milk”, “bread”
* Grocery\_list.pop(2)- index of item you want to remove

Would bread because it is at position 2

3/3/25 Nested while Loops and Lists

h

**AFTER QUIZ 2**

3/17

Test driven function-writing: before writing a function its helpful to focus on concrete examples on how the function should behave

* What are some usual arguments and expected return values?
  + Use cases or expected cases
* What are some valid but unusual arguments and expected return values?
  + Edge cases
  + Ex: incorrect inputs, empty inputs

We can write functions that test the correctness of other functions

* This is called testing

3/19 Sets and dictionaries

Enter: sets

Sets like llists are a data structure for storing collections of values

Unlike lists sets are unordered and each value has to be unique

Lists: always zero-based, sequential, integer indices

Benefit of sets: testing for the existence of an item takes only one “operation” regardless of the sets size.

pids: set[str] = {730120710, 730234567, 730000000}

Enter: Dictionaries

Lists are a data structure for storing collections of values.

* Unlike lists, dictionaries give you the ability to decide what to index your data by.
  + Lists: always zero-based, sequential, integer indices.

Dictionaries are indexed by keys associated with values. This is a unique one-way mapping!

* Analogous: a real-world dictionary’s keys are words, and associated values are definitions.

A screenshot of a computer program

Description automatically generated

3/21/25

3/24/2025

Sets:

Adding elements to a dictionary:

* <dict name>[<key>] = <value>
* Temps[“DC”] = 52.1

Access + modify

* To access a value use subscription notation
* <dict name>[key]
* To modify, also use subscription notation:
* <dict name>[<key>] = new\_value

Check if key in dictionary:

* <key> in <dict name>

Removing elements: use pop()

* <dict name>.pop(<key>)

“for” Loops

* Loops iterate over the keys by default
  + For key in ice\_cream:
    - Print(key)

3/26 REVIEW

How to add something to a set:

* <name of set**>.add**(<what you are adding>)

3/31 After quiz 3

Objects in python

Every object has a type and an internal data representation

Class: allows us to create our own data type in python

* **Class** Profile:
  + Username: str
  + Bio: str
  + Followers: int
  + Following: int
* Def \_init\_\_(self):
  + Self.username = “usr9”
  + Slef.bio = “”

Not a function but a method

My\_prof: profile = Profile()

My\_prof.username

2/4

Review of classes and objects

A class is like a blueprint/template

* Defines attributes and behaviors its objects will have

An object is an instance of a class

* If the objects is the blueprint, the object is the house
* Has all the specified attributes and behaviors.
* Different objects share these attributes and behaviors but are distinct.

Objects are a data abstraction

They all have

1. An internal representation
   1. Data attributes
2. An interface for interacting with the object
   1. Interface defines behaviors but hides implementation (the details)
   2. Methods: functions defined within a class
      1. Self is the first parameter

4/7 Magic Methods and Recursion Review

Stack overflow and recursion errors:

* When a program writes a function that calls itself indefinitely the function stack will overflow
  + Leads to a **overflow** or **Recursion Error**

Recursive function checklist:

Base Case:

* Does the function have a clear base case
* Will the base case be reached

Recursive case:

* Does the function have a recursive case that progresses toward the base case?
  + Ensure the bases case returns a result directly (without calling the function again)
* Have you tested your function with multiple cases, including edge cases?

4/9/25 Recursive structures and processes

**Anatomy of a singly linked list**

Node : going to have two attributes: 1 being a value and 2. One called next

* This node can refer to another node or can have the value of None

This node will also have two objects: 1 being a value and the other referring to the first node



In node statements the verticle bar means “or”

* Next: Node | Non