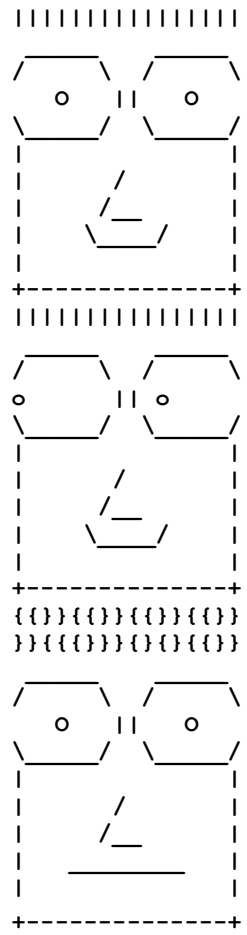
Learning Goals

Understand differences and similarities
Function definitions vs. function calls
Functions with return statements vs. those without
Functions with parameters vs. those without
Functions can be arguments
Be creative and learn lesson(s) about software design and engineering
Create a small, working program, make incremental improvements.
Read the directions and understand specifications

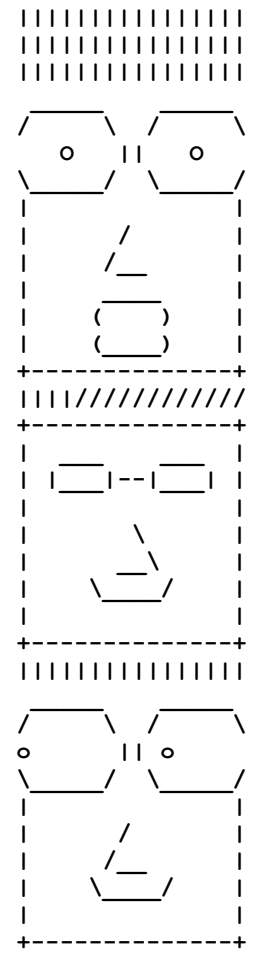
Introduction and Overview

In this assignment, you will design groups of faces (we may refer to a face also as a head) by creating functions that return strings and functions that print strings. Some functions will print an entire "face" and others will return strings representing parts of a face like eyes, chin, nose, hair, and so on. You'll also write functions with parameters that are functions!

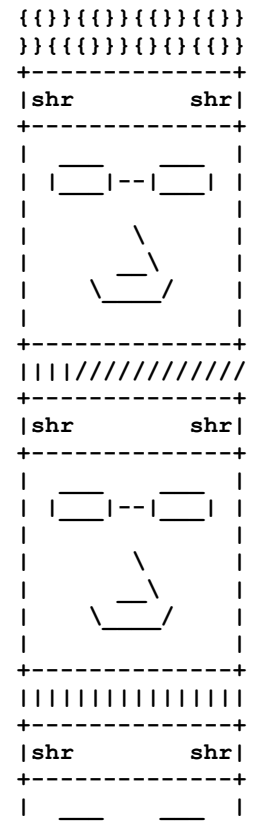
Three examples of groups of faces are shown next. These faces adhere to the requirements that are provided below.

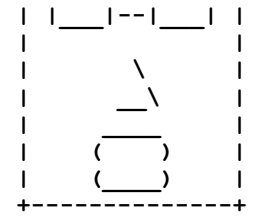


Here is a second group:



Here is a third group





There are no requirements on what your group of faces look like, but there are requirements on the number of characters in each line you print (18 characters), the names of the functions, and the three groups of faces your program must print.

This assignment will help you gain a better understanding of writing/calling functions, the difference between returning and printing, and the importance of adhering to design standards. It is also designed so that you can demonstrate some creativity, which is one of the important aspects of creating computational artifacts.

General Programming Tasks to Complete
You'll create a project following the steps outlined in the project creation details section below. In the main block you will call three functions, and each function will print one group of three faces:

The function faces_fixed() prints the same group of three faces each time it's called. The "fixed" indicates that the faces doesn't change.
The function faces_selfie() prints a group of three faces in which each face has a selfie band someplace in the face (top, middle, bottom).
The function faces_random() prints a possibly different group of three faces each time the program is run. The function does this by choosing characteristics (such as which mouth or which eyes) to draw based on a random number as described below.

How to Complete The Program
You should not write the entire program and then run and test it. Programs are best written incrementally; for example, rather than writing all 12 (or more) face part functions at once, you could first write enough of these to make one whole face function work, and then move on to the others. In general, creating a program that works by doing something, and then adding to the working program, is much more efficient in terms of programming time than writing the entire program at once and then testing it. Start small, make the small program work, then make the program more complete.

Functions You Must Write
In total you'll need to design, implement, and test about 25 functions as described here. As stated above, don't write all 25 before you run and test your program. You must follow

specific naming conventions and use underscores "_" to separate the words in your function names. The naming conventions are further described below with examples:

You'll write 12 or more face part functions that each return a string. These functions must be named using the convention part_DESCRIPTION, e.g., part_mouth_smile, or part_eyes_glasses, or part_hair_pointy. More details are given below.
You'll write three or more whole face functions without parameters that each print a complete face by calling some face part functions and printing the strings returned. These functions should have no parameters and should be named DESCRIPTION_face, e.g., surprised_face, or glasses_face.
You'll write three or more whole face functions that each have one or two parameters. Each parameter is the name of a face part function that will be called. These should be named face_with_DESCRIPTION, e.g., face_with_mouth, or face_with_hair, or face_with_multi.
You'll write three faces functions as described above at the beginning of this section: faces_fixed, faces_selfie, faces_random. Each of these functions calls three of the whole face functions to print a group of three faces.
You'll need to write a helper function named selfie_band. (You can also write other helper functions that might be useful in creating faces. See below for more examples.)
You'll need to write a helper function named face_random that uses the random module and the function random.randint to print a different face each time it's called. The face_random function should make use of one of your whole face functions with parameters, and it must be called from within faces_random.

Reminders/Tips Before Starting
Work through the assignment in the order that the instructions are given.
Pay strict attention to the specifications for names and return values in the assignment.
Document your code! Every function you write must have a docstring comment describing what it does. This includes any helper functions you write that are not explicitly asked for in these instructions. It will be easier to do this if you write comments as you write your functions, rather than having to go back and remember the purpose of each function after finishing the assignment. It is recommended to keep docstrings around 80 characters. Be creative and have fun!

Completing the Assignment
Create a new Project in your Java/Summer Camp MIL folder. In your project, create a new Python File called Faces.py:
The new module should contain a comment with the date and author name and

First, copy/paste or retype the main program block given below:

```
if __name__ == '__main__':
    print("\nfixed group of three faces\n")
    faces_fixed()

    print("\ngroup of three self faces\n")
```

```
faces_selfie()

print("\ngroup of three random faces\n")
faces_random()
```

In order to get this to run, you'll need to define the three functions that are called from the main block. For now, as a test, make the body of each function a single print statement, such as print("faces fixed"). Then you can run your program to see that it works. Once your program runs with the three faces functions each printing a single line, continue with the rest of the assignment.

Completing the Functions

Face Part Functions

Each face part function should return a string that could be printed to be part of a face (e.g. hat, hair, ears, eyes, nose, mouth, chin, beard, etc.). You must create a minimum of 12 face part functions, but we suggest you write just enough to create one face and test them before moving on. You must create functions of at least three different kinds/types, (e.g. nose, hair, mouth, eyes, etc.). You must create at least three functions for three of these types (e.g. three nose functions, three hair functions, and three mouth functions). As long as three different parts each have at least three choices, you can have more than three functions for some face parts. Make sure to keep in mind these requirements:

The face part functions must be named as specified: the word “part”, an underscore, the part of the face, an underscore, and a description, e.g. part_eyes_glasses or part_chin_pointed.
Each face part function must return a string exactly 18 characters long where both the first and last characters are spaces. You can include other space characters within the string returned.
Each function needs a docstring comment between triple quotes at the beginning of the function. The docstrings for the face part functions can be simple (see examples below).
You should use raw strings-- that is, the letter r should precede the first quote of the string. This will allow you to use backslashes easily in your returned strings. (See the note below for more about raw strings.)
Here are two examples that illustrate face part functions that adhere to these specifications:

```
def part_hair_pointy():
    """
    Returns a string that is
    pointy hair
    """
    a1 = r"012345678901234567"
    a2 = r" /\/\/\/\/\/\/\ "
    return a2

def part_nose_up():
    """
    Returns a string that is
    an upturned nose
```

```
    """
    a  = r"012345678901234567"
    a  = r" |       /\       | " + "\n"
    a += r" |      /  \      | "
    return a
```

You'll see in part_hair_pointy that a variable a1 is created but not returned. For part_nose_up, the variable a is first assigned the same string, but then it's reassigned to the first string of the nose. This string simply allows you to construct the other strings that will be returned so that each string has the right format: having spaces for the first and last characters and being exactly 18 characters long (including these spaces).

As the example functions show, the string returned can span more than one line. Simply concatenate each string and separate them with a newline character "\n", except for the last line.

Whole Face Functions Without Parameters

You must write at least three whole face functions without parameters as explained above. Rather than writing all three at once, we suggest writing one and testing it before writing more. Remember:

Each function must be named DESCRIPTION_face.
Each function must take no parameters.
Each function must call face part functions and print their return value.
Each function must have a docstring comment at the beginning.
Here's an example:

```
def funny_face():
    """
    Print a face that looks a little funny,
    With surprised mouth and eyebrows
    """
    print(part_hair_plain())
    print(part_eyes_withbrows())
    print(part_nose_big())
    print(part_mouth_surprised())
    print(part_chin_plain())
```

faces_fixed()

The function faces_fixed() simply prints three faces, one after another. As a test, you can call a single face function three times, e.g. the funny_face() function shown above. As you complete more face functions, you should call three different functions within faces_fixed(). Be sure to include a docstring comment.

```
def faces_fixed():
    """
    Print the same group of faces with three
    faces, one funny, one this, one the other
    """
```

```
funny_face()
funny_face()
funny_face()
```

At this point, you can test that your code prints out a group of three faces by running faces.py.

If you've been working incrementally as we recommend, you should go back and add some more face part functions and at least one more whole face function at this point.

Whole Face Functions With Parameters

For the whole face functions with parameters, each parameter will be the name of a function for one type of face part, e.g. mouth or eyes. Functions can be parameters in Python! You'll need to be careful to pass the name of the function, and not the return value. Here's an overview of the requirements and an example:

Name the function face_with_DESCRIPTION, e.g., face_with_mouth.
Name the parameter mouthfunc or eyefunc, or more generally partfunc.
Create a face by calling face part functions and printing the results, just as you did earlier.
One of the function calls will be the parameter as shown below where parameter mouthfunc is called in the body of face_with_mouth.

```
def face_with_mouth(mouthfunc):
    """
    Print a face with eyebrows and a big nose,
    but with a mouth specified by mouthfunc
    """
    print(part_hair_plain())
    print(part_eyes_withbrows())
    print(part_nose_big())
    print(mouthfunc())
    print(part_chin_plain())
```

You'll call the face_with_mouth function by passing a function name as follows, e.g. to create three different faces that have the same features except for the mouth:

```
face_with_mouth(part_mouth_surprised)
face_with_mouth(part_mouth_smile)
face_with_mouth(part_mouth_frown)
```

Things to keep in mind about using functions as parameters:

The parameter, e.g., mouthfunc, is the name of a function. You call it in the body of the face_with_x function like you call any function: by adding parentheses to the function name.

When you call the face_with_x function, you pass the name of a face part function. You do not call this face part function when passing it as a parameter! You pass its name.
Be careful, because when you write the function to pass, e.g., part_mouth_smile, Pycharm may autocomplete the parentheses. If you call the function part_mouth_smile when passing it as a parameter, e.g.,
face_with_mouth(part_mouth_smile()) # Don't do this!

then you'll be passing the return value of part_mouth_smile(), a string, rather than passing the name of the function. This will generate a Type error: 'str' is not callable because the code in face_with_x will try to call the string passed as if it's a function.

You'll need to implement three whole face functions with parameters. You can optionally have one function which takes more than one parameter, e.g. the name of a mouth function and the name of a hair function. Be sure to call one of your whole face functions with parameters from inside face_random() as described below so that your random group of faces includes one of these faces. (Feel free to use the remaining two of these functions in either of the other groups of faces.)

faces_random() and face_random()
For the faces_random()function, you'll need a helper function named face_random(). The face_random() helper function will generate a different face each time it's called. Your program will need to generate random numbers using the random module and then use these numbers to specify parameters for function calls. face_random() must be called in the function body of faces_random() as shown:

```
def faces_random():
    """
    Print three random faces
    using face_random()
    """
    face_random()
    face_random()
    face_random()
```

At the top of Faces.py (under the module docstring) you'll need to import the random module, e.g. you'll write

import random

You'll need to call random.randint(low,high), e.g. random.randint(1,4) to generate a random integer between low and high (inclusive). Store this value in a variable and use it to generate different faces by choosing face part functions to pass to one of your whole face functions with parameters. Here is an example of how to use a random value to create different calls to a whole face function that takes parameters:

```
def face_random():
```

```
"""
Print a face with randomly chosen
mouth and eyes
"""
eyefunc = part_eye_glasses
mouthfunc = part_mouth_open
x = random.randint(1,3)
if x == 1:
    mouthfunc = part_mouth_frown
elif x == 2:
    eyefunc = part_eye_withbrows
else:
    mouthfunc = part_mouth_surprised
    eyefunc = part_eye_closed
# now call the function
face_with_two(eyefunc,mouthfunc)
```

Now run Faces.py multiple times. Each time you run it, the first two groups of faces should remain the same but the random group of faces should change.