Kids Hack Labs

Pseudocode Standards

You are not expected to be an expert at coding, but you should know and be able to follow basic instructions when it comes to programming in Python. The purpose of this document is to guide you through the “basic building blocks” of translating pseudocode into actual code.

Because this is a guide, you are encouraged to understand it and apply the concepts to your needs. **Do not** copy them verbatim: chances of your code not working are high if you don’t adapt the instructions to the context of what you are coding. You are expected to use this guide during coding time until a coach is able to help you.

On Basic Python

Instruction	**Meaning**	**Code examples**

Create a variable to hold the name of our player
 	
Create a variable to track whether our player is alive. Initialize it to True
 	
Create a variable to track the player’s score
 	
Create a variable to keep track of player lives. Initialize it to 3
	This means that the variable has a predefined type for us to work with. In this case, give it a meaningful name and initialize it to a neutral value of that type. **Unless stated otherwise**, ints and floats are initialized to zero (`0`), booleans are initialized to `False`, and strings are initialized to empty strings(`""`). The context should be enough for you to know what type you will need to use when you initialize our variable.	`player_name = ""`
		
		`player_alive = True`
		
		`is_alive = True`
		
		`player_score = 0`
		
		`player_lives = 3`

Create a list/tuple to store our list of prime numbers up to 10	This means that the variable will be assigned a list or a tuple with multiple items. The sequence’s name should be meaningful	`primes_list = [2, 3, 5, 7]`
		
		`primes_tuple = (2, 3, 5, 7)`

Create a list/tuple with four elements to store the special pellet locations	If a sequence has a set number of elements, you may have to initialize all of them to "neutral" values of the appropriate type. When in doubt, check your own code for other examples.	`special_pellet_location = ((0, 2),(20, 2),(),())`
Create a dictionary to store our items. Keys should be the items' names, and values should be the item's prices	This is a specific instruction on how to create a dictionary. Based on the context, you can assume that, in this case, the dictionary created has strings as keys and ints or floats as values. Remember to give your dictionary a meaningful name	`inventory = {"Boots": 50, "Bottle":2, "Eggs": 5, "Sand" = 10}`

Read/access the third item of the primes list/tuple.
 	
(coach verbal instruction) prime list/tuple at three
	This isn't as straightforward, considering lists are 0-indexed in Python. The third item is actually index 2. Notice that the instruction will also mention the contents of the list, not necessarily its name. This means that you are the one responsible for keeping track of the names you give to your variables	`primes_list[2]`
		
		`primes_tuple[3]`

Create a function to move the player character	Straightforward function creation by using the keyword **`def`** followed by the function's name.
	
	Unless stated otherwise, you're free to choose the function's name, but it should still be a meaningful one.
		`def move_character():`
		
		`def move_player():`

Create a function that takes three arguments as inputs and prints the result on the command line	The function should have a meaningful name and account for the number of arguments. As this function is expected to print something to the screen, the call to Python's print() function should be done inside the function as well	`def print_sum(a, b, c):` `    print(a+b+c)`
Create a function that takes two arguments and returns/outputs whether the first one is divisible by the second one	The function should have a meaningful name, account for the number of arguments **and** a `return` statement	`def is_divisible(a, b):` `    return a%b == 0`

If our timer has reached zero	This means that we need an if statement. **Depending on the context** of what we're writing, the statement needs to use the proper comparison operators.	`if timer > 0:`
		`if timer < 0:`
		`if timer >= 0:`
		`if timer <= 0:`

Create a for loop that iterates over our game_object list For each object in our game_object list	The for loop can take one of two forms, depending on our needs: if we just need each game_object we can use a "`for obj in list:`" loop. If we need the object's indices, we can use a "`for i in range(len(list)):`" syntax. Notice the double brackets at the end of the second type of loop.	`for obj in game_objects:` `for i in range(len(game_objects)):`
Create a loop that repeats 20 times	Straightforward "`for i in range():`" loop	`for i in range(20):`

Create a loop that repeats while our timer has not reached zero:	Again, the context needs to be taken into account. Four possibilities are available:	`while timer > 0:`
		`while timer < 0:`
		`while timer >= 0:`
		`while timer <= 0:`

Call my_function()
 	
Call the set_position() function with the coordinates 43, 125
 	
Call the update_timer() function. Do not forget to pass delta as an argument
	Straightforward use of functions. Notice that you are expected to **use** the function, and **not create** it. If the function has arguments, those should be passed as well. Unless the values are explicit, variable names should be used instead	`my_function()`
		
		`set_position((43, 125))`
		
		`update_timer(delta)`

Import the random module into your code	Straightforward import module	`import random`
Import the X class from the Y module	Straightforward `from X import Y` statement.	`from pygame import Color, Rect, Surface`
Import the circle function from pygame's draw submodule	Pay attention to the relationships between the modules and submodules!	`from pygame.draw import circle`

Import the A and B constants from the Z module	Here, we're importing specific values from a module. We keep our application small: import only what's needed. This is similar to importing specific functions; the instructions will often be specific about the imports.	`from pygame.locals import K_UP, K_DOWN`
		
		`from math import PI`

Add y to x
 	
Subtract y from x
 	
Multiply x by y
 	
Divide x by y
 	
Assign x the result of a division of y by x
	This means that, in addition to doing the mathematical operation, you need to assign the result back to the first variable/attribute. This makes sense in context: in coding, there's very little reason to do math without storing the results of your calculations somewhere.
	
	Pay attention to the order in which the variables are mentioned in the instruction so you don't get it wrong when coding. This is especially important in subtractions and divisions where x-y is different from y-x.
		`x += y`
		
		`x = x - y`
		
		`x *= y`
		
		`x /= y`
		
		`x = y/x`

On Not-so-Basic Python

Instruction	**Meaning**	**Code examples**
Create a class to manage the player's movement	Straightforward class creation instructions. As per Python guidelines, class names must begin with Capital letters. If a name has more than one component, the first letter of each component must be Capitalized as well. Remember: **no spaces** between component names.	`class PlayerMovement():`
Create the `Scene` class	If the instructions tell you to create a class with a specific name, follow them. There are likely engine-based reasons for that, and your game will not work if you use a different name. Unless stated otherwise, the class should have the same name as the file, but with the first letter capitalized	`class Scene():`

Create an attribute to store our player's number of lives and initialize it to zero
 	
Create an attribute to store our player's name
	Attributes must have a meaningful name.
	
	Attributes must be of the correct type, and initialized properly.
	
	Unless otherwise noted, attributes must be preceded by the `self.`. notation.
		`self.lives = 3`
		
		`self.name`

Create the X class's initializer	Unless noted otherwise, the only needed argument is **`self`**. Mind the space between "**`def`**" and the first underscore.	**`def __init__(self):`**
Create the X class's check_collision() method. It should take two arguments aside from self: the player position and the pellet position	You are strongly recommended to use the given name. Remember that the argument list begins with **`self`**, followed by the others that may be indicated in the instructions.	**`def check_collision(self, player_pos, pellet_pos):`**
Import the X class from the source folder's Y file/module	Python "understands" files and folders more or less in the same way; that is, "files", "folders", and "modules" are "the same thing" for the Python environment. Because of this, the dot notation can be used just like it's used for submodules. Be sure to use the class's correct name and spelling	**`from src.collectible import Collectible`**

Coding terms:

Application main entry point: the **`main()`** function. It is usually a short function that simply creates an object of the application's class and runs it. Example:

```
from game import Game
def main():
    g = Game()
    g.run()
```

Asset: this refers mostly to artwork-related files, namely, images and sounds. They are usually stored in the **`assets`** folder.

Driver code: Python is an interpreted language. This means that the code actually runs on top of another program (that is, the shell). To run "our" program, we need to specifically tell the computer to do so. The driver code usually looks like this:

```
if __name__ == "__main__":
    main()
```

Notice that this code usually goes at the bottom of the main.py file (where the main function is defined).

Filepaths and filenames: Whenever the code needs a **filepath**, it means that you should provide the **`string`** that matches the path to the file you want/need to load. For example, **`pygame.image.load()`** requires a **filepath** and a **filename** to be able to load an image into our game in a format such as **filepath** + **filename**. The code should look something like this:

```
pygame.image.load("./assets/filename.png")
```

Folder structure: Take some time to familiarize yourself with our project's folder structure. This means that you should know where our code files and where our asset files are stored. Our game shouldn't have many files, and they should be stored mainly in the **`src`** (short for "source code") and **`assets`** (images, sounds) folders.

Quotient: the mathematical concept of the "whole part" of a division between two integer numbers. In Python, this is obtained by using the // operator. For example, the quotient of seven by two is three:

```
>>>7//2
3
```

This kind of code can appear on the right-hand side of an assignment, meaning that you can store the result of **`7//2`** in a variable.