

# 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
<p>Create a variable to hold the name of our player</p> <p>Create a variable to track whether our player is alive. Initialize it to True</p> <p>Create a variable to track the player's score</p> <p>Create a variable to keep track of player lives. Initialize it to 3</p>	<p>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. <b>Unless stated otherwise</b>, ints and floats are initialized to zero (0), booleans are initialized to <b>False</b>, 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.</p>	<pre>player_name = ""  player_alive = True is_alive = True  player_score = 0  player_lives = 3</pre>
<p>Create a list/tuple to store our list of prime numbers up to 10</p>	<p>This means that the variable will be assigned a list or a tuple with multiple items. The sequence's name should be meaningful</p>	<pre>primes_list = [2, 3, 5, 7] primes_tuple = (2, 3, 5, 7)</pre>

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.	<code>special_pellet_location = ((0, 2), (20, 2), (), ())</code>
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	<code>inventory = {"Boots": 50, "Bottle": 2, "Eggs": 5, "Sand" = 10}</code>
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	<code>primes_list[2]</code>  <code>primes_tuple[3]</code>
Create a function to move the player character	Straightforward function creation by using the keyword <code>def</code> 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.	<code>def move_character():</code>  <code>def move_player():</code>

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	<pre>def print_sum(a, b, c):     print(a+b+c)</pre>
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 <b>and</b> a <b>return</b> statement	<pre>def is_divisible(a, b):     return a%b == 0</pre>
If our timer has reached zero	This means that we need an if statement. <b>Depending on the context</b> of what we're writing, the statement needs to use the proper comparison operators.	<pre>if timer &gt; 0: if timer &lt; 0: if timer &gt;= 0: if timer &lt;= 0:</pre>
<p>Create a for loop that iterates over our game_object list</p> <p>For each object in our game_object list</p>	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.	<pre>for obj in game_objects: for i in range(len(game_objects)):</pre>
Create a loop that repeats 20 times	Straightforward "for i in range():" loop	<pre>for i in range(20):</pre>

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:	<pre>while timer &gt; 0: while timer &lt; 0: while timer &gt;= 0: while timer &lt;= 0:</pre>
<p>Call my_function()</p> <p>Call the set_position() function with the coordinates 43, 125</p> <p>Call the update_timer() function. Do not forget to pass delta as an argument</p>	<p>Straightforward use of functions. Notice that you are expected to <b>use</b> the function, and <b>not create</b> it. If the function has arguments, those should be passed as well. Unless the values are explicit, variable names should be used instead</p>	<pre>my_function()  set_position((43, 125))  update_timer(delta)</pre>
Import the random module into your code	Straightforward import module	<pre>import random</pre>
Import the X class from the Y module	Straightforward <b>from X import Y</b> statement.	<pre>from pygame import Color, Rect, Surface</pre>
Import the circle function from pygame's draw submodule	Pay attention to the relationships between the modules and submodules!	<pre>from pygame.draw import circle</pre>

# 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: <b>no spaces</b> between component names.	<code>class PlayerMovement() :</code>
Create the <b>Scene</b> 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	<code>class Scene() :</code>
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 <b>self..</b> notation.	<code>self.lives = 3</code>  <code>self.name</code>

Create the X class's initializer	Unless noted otherwise, the only needed argument is <b>self</b> . Mind the space between “ <b>def</b> ” and the first underscore.	<b>def __init__(self):</b>
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 <b>self</b> , followed by the others that may be indicated in the instructions.	<b>def check_collision(self, player_pos, pellet_pos):</b>
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	<b>from src.collectible import Collectible</b>

## 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()
```

**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).