

The Create Task

Create Task




- Written in Python
- Processing or replit
- Game, application, or mathematical simulation.

Programming is a collaborative and creative process that brings ideas to life through the development of software. In the Create performance task, you will design and implement a program that might solve a problem, enable innovation, explore personal interests, or express creativity. Your submission must include the elements listed in the Submission Requirements section below.

You are allowed to collaborate with your partner(s) on the development of the program only. **The written response and the video that you submit for this performance task must be completed individually, without any collaboration with your partner(s) or anyone else.** You can develop the code segments used in the written responses (parts 3b and 3c) with your partner(s) or on your own during the administration of the performance task.

Note: Because of all of the citation requirements, I recommend that you work on this task individually. If the program you are writing requires help from others, consider something simpler!

Requirements

-  **Final program code** (created independently or collaboratively)
-  **A video that displays the running of your program and demonstrates functionality you developed** (created independently)
-  **Written responses to all the prompts in the performance task** (created independently)

1. PROGRAM CODE (CREATED INDEPENDENTLY OR COLLABORATIVELY)

Submit one PDF file that contains all of your program code (including comments). Include comments or acknowledgments for any part of the submitted program code that has been written by someone other than you and/or your collaborative partner(s).

IMPORTANT:

If the programming environment allows you to include comments, this is the preferred way to acknowledge and give credit to another author. However, if the programming environment does not allow you to include comments, you can add them in a document editor when you capture your program code for submission.

In your program, you must include student-developed program code that contains the following:

- Instructions for input from one of the following:
 - ◆ the user (including user actions that trigger events)
 - ◆ a device
 - ◆ an online data stream
 - ◆ a file

- Use of at least one **list** (or other **collection type**) to represent a collection of data that is stored and used to manage program complexity and help fulfill the program's purpose

IMPORTANT:

The data abstraction must make the program easier to develop (alternatives would be more complex) or easier to maintain (future changes to the size of the list would otherwise require significant modifications to the program code).

- ❑ At least one procedure that contributes to the program's intended purpose, where you have defined:
 - ◆ the procedure's name
 - ◆ the return type (if necessary)
 - ◆ one or more parameters

IMPORTANT:

Implementation of built-in or existing procedures or language structures, such as event handlers or main methods, are not considered student-developed.

- ❑ Calls to your student-developed procedure
- ❑ Instructions for output (tactile, audible, visual, or textual) based on input and program functionality
- ❑ An algorithm that includes sequencing, selection, and iteration that is in the body of the selected procedure

2. VIDEO (CREATED INDEPENDENTLY)

Submit one video file that demonstrates the running of your program as described below. Collaboration is **not** allowed during the development of your video.

Your video must demonstrate your program running, including:

- ☐ Input to your program
- ☐ At least one aspect of the functionality of your program
- ☐ Output produced by your program

Your video may NOT contain:

- ☐ Any distinguishing information about yourself
- ☐ Voice narration (though text captions are encouraged)

Your video must be:

- ☐ Either .mp4, .wmv, .avi, or .mov format
- ☐ No more than 1 minute in length
- ☐ No more than 30MB in file size

3. WRITTEN RESPONSES (CREATED INDEPENDENTLY)

Submit your responses to prompts 3a – 3d, which are described below. Your response to all prompts combined must not exceed 750 words (program code is not included in the word count). Collaboration is **not** allowed on the written responses. Instructions for submitting your written responses are available on the [AP Computer Science Principles Exam Page](#) on AP Central.

3 a. Provide a written response that does all three of the following:

Approx. 150 words (for all subparts of 3a combined)

i. Describes the overall purpose of the program

ii. Describes what functionality of the program is demonstrated in the video

iii. Describes the input and output of the program demonstrated in the video

- 3b.** Capture and paste two program code segments you developed during the administration of this task that contain a list (or other collection type) being used to manage complexity in your program.

Approx. 200 words (for all subparts of 3b combined, exclusive of program code)

- i.** The first program code segment must show how data have been stored in the list.



- ii.** The second program code segment must show the data in the same list being used, such as creating new data from the existing data or accessing multiple elements in the list, as part of fulfilling the program's purpose.



3b continued

Then, provide a written response that does all three of the following:

- iii. Identifies the name of the list being used in this response

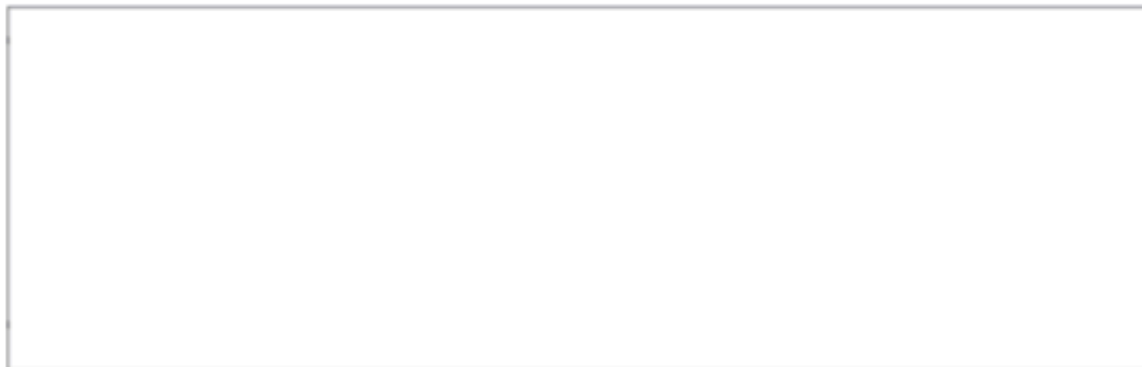
- iv. Describes what the data contained in the list represent in your program

- v. Explains how the selected list manages complexity in your program code by explaining why your program code could not be written, or how it would be written differently, if you did not use the list

- 3c.** Capture and paste two program code segments you developed during the administration of this task that contain a student-developed procedure that implements an algorithm used in your program and a call to that procedure.

Approx. 200 words (for all subparts of 3c combined, exclusive of program code)

- i. The first program code segment must be a student-developed procedure that:
- ☐ Defines the procedure's name and return type (if necessary)
 - ☐ Contains and uses one or more parameters that have an effect on the functionality of the procedure
 - ☐ Implements an algorithm that includes sequencing, selection, and iteration

A large, empty rectangular box with a thin black border, intended for the student to paste their program code segments. It occupies the lower half of the page.

3c continued

- ii. The second program code segment must show where your student-developed procedure is being called in your program.

Then, provide a written response that does both of the following:

- iii. Describes in general what the identified procedure does and how it contributes to the overall functionality of the program

- iv. Explains in detailed steps how the algorithm implemented in the identified procedure works. Your explanation must be detailed enough for someone else to recreate it.

3d. Provide a written response that does all three of the following:

Approx. 200 words (for all subparts of 3d combined)

- i. Describes two calls to the procedure identified in written response 3c. Each call must pass a different argument(s) that causes a different segment of code in the algorithm to execute.

First call:

Second call:

- ii. Describes what condition(s) is being tested by each call to the procedure

Condition(s) tested by the first call:

Condition(s) tested by the second call:

- iii. Identifies the result of each call

Result of the first call:

Result of the second call:

AP Computer Science Principles Policy on Plagiarism

The use of media (e.g., video, images, sound), data, information, evidence, or program code created by someone else in the creation of a program and/or a program code segment(s), without appropriate acknowledgment (i.e., through citation, through attribution, and/or by reference), is considered **plagiarism**. A student who commits plagiarism will receive a score of 0 on the performance task.

To the best of their ability, teachers will ensure that students understand how to ethically use and acknowledge the ideas and work of others, as well as the consequences of plagiarism. The student's individual voice should be clearly evident, and the ideas of others must be acknowledged, attributed, and/or cited.

During the final submission process in the AP Digital Portfolio, students will be asked to attest that they have followed the performance task guidelines and have not plagiarized their submission.

If there is a game that you have always wanted to write, DON'T do it for the create task!

The create task should be as minimal as possible, satisfying all of the requirements and tailored to answer the free response questions!

As we'll see from the examples this week, it can be a very simple program(see the replit example [Substitution Encryption](#)).And it's ok if it's a boring program!

It's not the program that gives you a high score; it's the free response answers!
A student in the past wrote Tetris and only got a 3!

You can use:

1) replit: **This is a highly recommended option for most students.** This will limit your create task to something simple without all of the debugging errors that come with Processing.

2) **Processing: I don't recommend this option.**

Students end up spending too much time on getting the program to work and less time on the free response answers and often score low on the task.

Use only this option if you have done all of the labs and were able to get them to work without too much help. In fact, you'll need permission from me if you like to use Processing for your create task.

Create Task Example I

We will walk through a full example of a Create Task. In this example, we will do a simple encryption program. Given a message **plaintext**, we like to encrypted into a secret message called **ciphertext**.

We do this by simply substituting each letter with a different letter from a secret **key** which is some permutation of the alphabet.

Sample key: YTNSHKVEFXRBAUQZCLWDMIPGJO

In this case, 'A' is replaced with 'Y', 'B' is replaced with 'T', etc...

This substitution is case sensitive 'a' is replaced with 'y', 'b' with 't', etc...

Any character not in the key is kept the same.

Create Task Example I

Sample key: YTNSHKVEFXRBAUQZCLWDMIPGJO

If the plaintext is 'cat', the ciphertext is 'nyd'.

If the plaintext is 'Cat', the ciphertext is 'Nyd'.

If the plaintext is 'Cat123', the ciphertext is 'Nyd123'.

Create Task Example I

Some preliminaries:

`ord()` converts a single Unicode character to its integer code point.

```
In [1]: ord('A')
```

```
Out [1]: 65
```

```
In [2]: ord('a')
```

```
Out [2]: 97
```

```
In [2]: ord('😊')
```

```
Out [2]: 128512
```

Create Task Example I

```
key = "YTNSHKVEFXRBAUQZCLWDMIPGJO"  
letter = 'D'  
index = ord(letter) - ord('A')  
cipherletter = key[index]
```

The full code for this task can be found [here](#).

Example 2(Optional)

This is a second example of a create task. In this example, the input to the program will be a text file. In the previous example, our list was a list of letters that represent some key.

Suppose we are writing a program that manages a database of students. Our list will now store a collection of students. We need a data type that contains information about a student.

A student has more than just one piece of information: name, age, id, address, etc... These collectively are the **data** of a student.

A student might have some **functionality**: ability to print personal information, change their address, update school information.

How do we store such complex information about a student?

Example 2(Optional)

We have seen this before in Processing. The class `Sprite` was able to store many pieces of data about a `Sprite`: position, velocity, image. It also stored different functionality of what a `Sprite` can do: draw itself, update itself, get left side of `Sprite`, set left side of `Sprite`, etc..

We like a data type that can bundle **data** and **functionality** into one variable. A **class** bundles together *data* (instance variables or attributes) and *functionality* (methods). Another name for class is **type**.

Motivated by the `Sprite` class, we will write the `Student` class. Compare this example to the `Sprite` class we have used in Processing.

An Example of a class(Optional)

main.py

```
class Student:
```

```
    def __init__(self, name, id):
```

```
        self.name = name
```

```
        self.id = id
```

```
    def print_info(self):
```

```
        print(self.name, self.id)
```

```
def main():
```

```
    s1 = Student("Mike Smith", 34323)
```

```
    s2 = Student("Sarah Jones", 67432)
```

```
    print(s1.name)      # Mike Smith
```

```
    print(s2.name)      # Sarah Jones
```

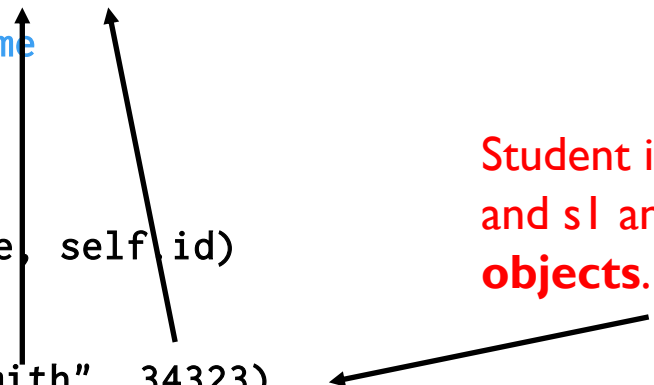
```
    print(s1.id)        # 34323
```

```
    s1.print_info()     # Mike Smith 34323
```

```
    s2.print_info()     # Sarah Jones 67432
```

```
main()
```

Student is a **class**(or **type**)
and s1 and s2 are two of its
objects.



(Optional)

This Create Task example will read in a text file of students: name and list of grades. It will create a list of Student objects and have provide some filtering capabilities.

<https://replit.com/@LongNguyen18/CreateTaskStudentGrades>

Steps for Create Task

- 1) Design Program: Brainstorm and think carefully about your program. What is your list? What is the student-developed procedure with parameters that used iteration(loops) and selection(conditionals)? Plan before writing code can save a lot of time!
- 2) Code the Program: Troubleshoot, test and refine. Start writing the student developed procedure and test it with hard-coded inputs. For example, from our encrypt example: `print(encrypt("hello", sub))`. Refine and repeat.
- 3) Make Video
- 4) Complete Written Response 3a.
- 5) Four Screenshots for 3b: i and ii, 3c: i and ii.
- 6) Complete written response 3b, 3c and 3d.
- 7) Submit portfolio on College Board site.

Timeframe

- 1) Design including brainstorming(2 hours)
- 2) Coding(5 hours)
- 3) Video(1 hour)
- 4) Written Response(2 Hours)
- 5) Review and refine written response answers using rubric(1 hour)
- 6) Uploading to Digital Portfolio(1 hour)

Total: 12 hours

This is the minimum amount of time you should spend on this. Likely you will need to spend more time(outside of class) to refine and create the best possible portfolio.

Program Code

Minimum Requirements:

- 1) Some type of input and output(input can be from keyboard or from file)
- 2) At least one list or dictionary that manages complexity.The list or dictionary should be essential to your program.Your program should be very difficult to write without the use of this list/dictionary.
- 3) At least one student-developed procedure that has an algorithm which uses one or more parameters that used iteration(loops) and selection(conditionals).

Program Code

The goal is to complete your code by April 16(day before April break).

There will be weekly check-ins. See Google classroom assignments.

First Check In(April 2): list and student-developed function, tested with hard-coded inputs.

Second Check-In(April 9): helper functions and refining program.

Third Check-In(April 16): main function and finishing up program. At the latest, program should be done after April break.

First Check-In Example:

list and student-developed function, tested with hard-coded inputs.

```
def encrypt(pt, sub):  
    result = ""  
    for c in pt:  
        if c.islower():  
            index = ord(c) - ord('a')  
            result += sub[index]  
        elif c.isupper():  
            index = ord(c) - ord('A')  
            result += sub[index].upper()  
        else:  
            result += c  
    return result
```

```
sub = ['y', 't', 'n', 's', 'h', 'k', 'v', 'e', 'f', 'x', 'r', 'b', 'a', 'u', 'q', 'z',  
       'c', 'l', 'w', 'd', 'm', 'i', 'p', 'g', 'j', 'o']  
print(encrypt("hello", sub))  
print(encrypt("Hello", sub))  
print(encrypt("Hello123", sub))
```


Second Check-In Example:

helper functions and refining program.

Previous code from last slide PLUS:

```
def main():
    # sample key: YTNSHKVEFXRBAUQZCLWDMIPGJO
    key = input("Enter 26-character key for encryption:")
    sub = create_substitution(key)
    plain = input("Enter plaintext: ")
    cipher = encrypt(plain, sub)
    print("cipertext: ", cipher)

main()

    exit("no duplicates!")
else:
    sub.append(k.lower())

return sub
```

Third Check-In Example:

adding main function and finishing up program

Previous code from last two slides PLUS:

```
def main():  
    # sample key: YTNSHKVEFXRBAUQZCLWDMIPGJO  
    key = input("Enter 26-character key for encryption:")  
    sub = create_substitution(key)  
    plain = input("Enter plaintext: ")  
    cipher = encrypt(plain, sub)  
    print("cipertext: ", cipher)  
  
main()
```