

## תרגיל 5: תכנות מונחה עצמים

תאריך פרסום: 27.12.2024

תאריך הגשה: 12.01.2025 בשעה 23:59

מתרגל אחראי: טל מלול

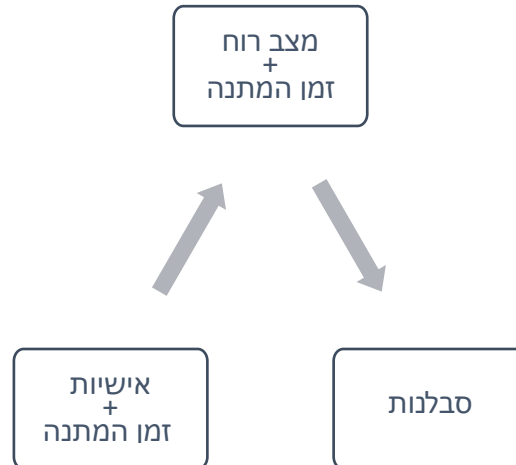
משקל תרגיל: 5 נקודות

### דגשים:

1. מומלץ לקרוא את כל התרגיל **לפני** המימוש. אנו ממליצים להבין היטב את ארכיטקטורת המערכת ולתכנן אותה על דף נייר לפני תחילת כתיבת הקוד.
  2. בניגוד לתרגילים קודמים, **בתרגיל זה אין להניח שהקלט תקין!** עבור כל שיטה / פונקציה מוגדרות הדרישות לגבי הקלט. הנחות על קלטים מסוימים יצוינו באופן מפורש. ובמידה והקלט לא תקין עליכם לזרוק חריגה מתאימה.
  3. מכיוון שהעבודה עמוסה מבחינת כמות הקוד שעליכם לממש וכדי להקל עליכם - אינכם נדרשים לכתוב docstring בעבודה זו.
  4. בתרגיל **לא מצוין** באופן מפורש איך עליכם לממש את הפתרון. לדוגמא, לא יצוין מתי עליכם לדרוס שיטה, להשתמש בשיטת האב, להגדיר מחלקה\שיטה אבסטרקטית, להגדיר שדה פרטי, לבצע shallow copy או deep copy וכדומה. עליכם יהיה להחליט על פרטי המימוש **תוך עמידה בדרישות התרגיל ובעקרונות שלמדתם בקורס** (לדוגמא, להימנע משכפול קוד).
  5. אין לשנות את השיטות המוגדרות בתרגיל. ניתן להוסיף **שיטות פרטיות** בלבד למחלקות הקיימות. מעבר לכך, אתם יכולים להשתמש בכל ספרייה פנימית של python.
- נא לקרוא את כל העבודה **לפני** שאתם מתחילים לכתוב את הקוד (זו הפעם השניה של הסעיף הזה ולא במקרה..).

## מלך הפלאפל: הקדמה

במטלה הזו תתבקשו לממש גרסה מצומצמת של המשחק "מלך הפלאפל". במשחק אתם מנהלים דוכן פלאפל והמטרה שלכם היא להרוויח כמה שיותר כסף על ידי מכירת מנות פלאפל המוזמנות ע"י הלקוחות. לכל לקוח מידת סבלנות מסוימת שקטנה בקצב שמושפע ממצב רוחו וזמן ההמתנה למנה שלו. לכל לקוח יש אישיות שמכתיבה איך מצב הרוח שלו משתנה בהתאם לזמן ההמתנה. אם ללקוח נגמרת הסבלנות, הוא עוזב את הדוכן. אם מספר הלקוחות שעזבו חורג מסף מסוים – המשחק מסתיים.



### חוקי המשחק:

בכל שלב, מגיעים לקוחות חדשים ומזמינים מנות פלאפל. עליכם להכין את המנה לפי ההזמנה ולהגיש אותה ללקוח לפני שהסבלנות שלו נגמרת.

### ניהול המשחק:

קיימים שני מנגנונים אפשריים לניהול הזמנות:

1. אקראית – הזמנות נוצרות באופן אקראי.
2. קבועה מראש – ההזמנות נוצרות באופן ידוע מראש.

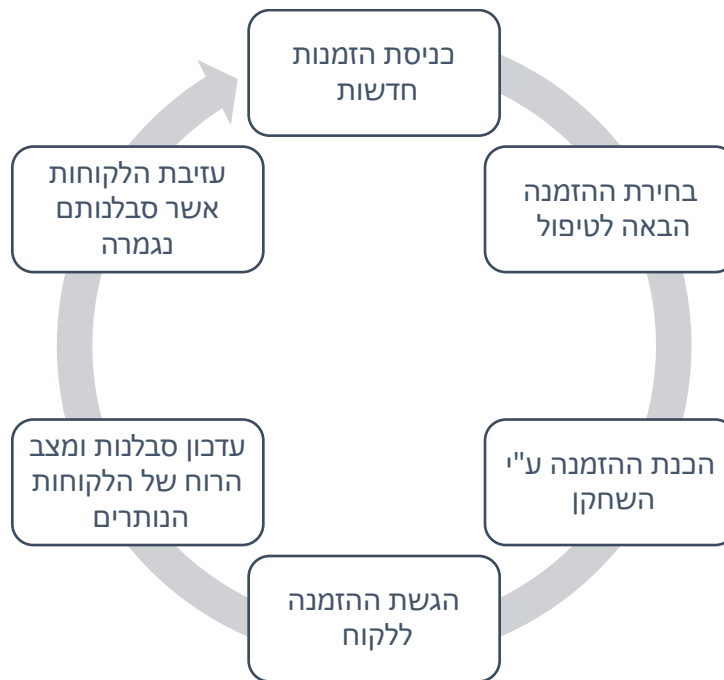
### בחירת ההזמנה הבאה לטיפול:

בכל נקודת זמן, עליכם להחליט איזו הזמנה להכין כעת. לצורך כך, תממשו מספר אסטרטגיות עבור ההחלטה מה תהיה ההזמנה הבאה לטיפול. למשל, אסטרטגיה אחת יכולה לתעדף הזמנות על פי סדר הגעה, אסטרטגיה שניה על פי זמן ההמתנה הארוך ביותר, ואסטרטגיה שלישית על פי סבלנותו של הלקוח.

### מהלך המשחק וסיומו:

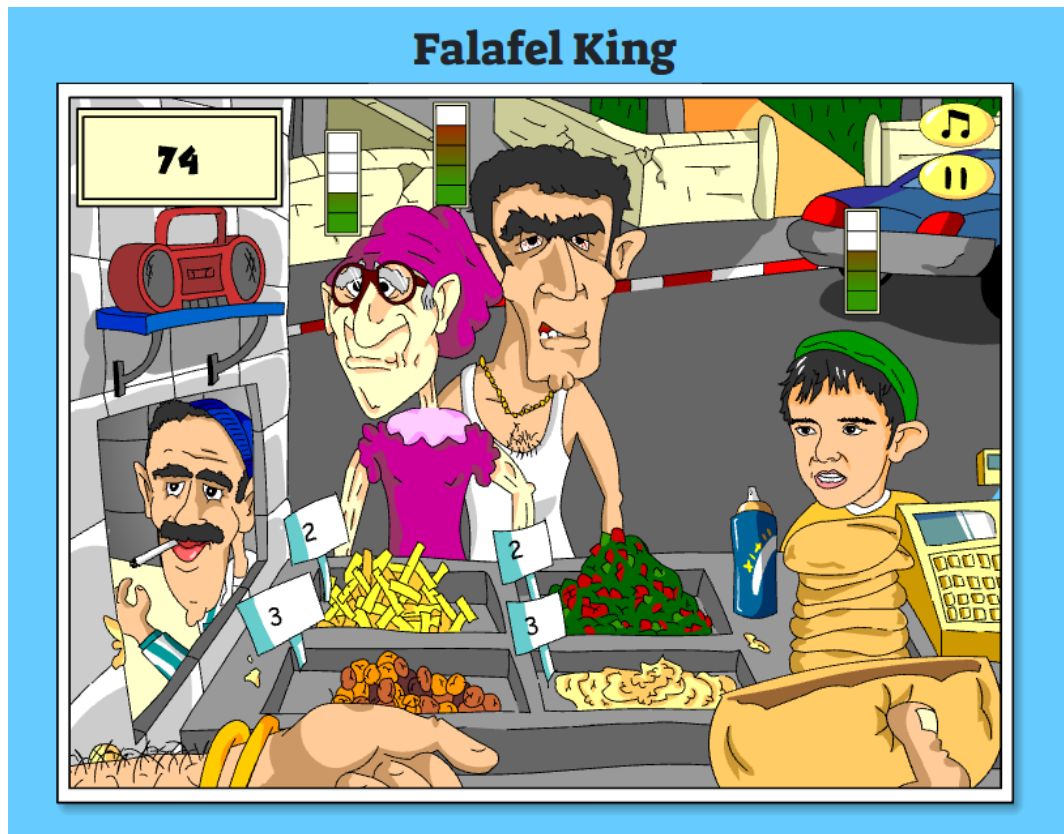
בכל שלב, לפי אחד ממנגנוני ניהול ההזמנות, מגיעים לקוחות חדשים ומזמינים מנות פלאפל. הזמנות אלו ייתוספו להזמנות שכבר קיימות בדוכן הפלאפל. לאחר מכן, לפי אחת מאסטרטגיות תעדוף ההזמנות, תיבחר הזמנה הבאה לטיפול. כעת, יוצג תפריט שיאפשר לכם להרכיב את ההזמנה באמצעות הבנסת קלט, להגיש ללקוח ולגבות תשלום עבור ההזמנה. בכל סיבוב, יש לעדכן את סבלנותם ומצב רוחם של הלקוחות שהזמנתם עדיין לא טופלה. אם סבלנותו של לקוח נגמרה, הוא יעזוב את הדוכן והזמנתו תוסר מרשימת ההזמנות. משחק מסתיים כאשר כמות הלקוחות שעזבו (כי נגמרה להם הסבלנות) תעבור סף מסוים. בסיום, יוצג הסכום שהרווחתם.

מצורפת תמונה המתארת את מהלך המשחק:



על מנת לממש את המשחק תתבקשו לתכנן ולממש מחלקות תוך יישום עקרונות תכנות מונחה עצמים (OOP) כפי שלמדתם בהרצאות ובתרגולים: אבסטרקציה – abstraction, הכמסה – encapsulation, ירושה – inheritance, רב-צורתיות – polymorphism.

בהצלחה!



למי שירצה ליהנות מהמשחק המקורי יכול למצוא אותו כאן: <https://www.falafelgame.com/>

## מבנה ההגשה

### שימו לב:

1. עליכם להגיש בעבודה זו 24 קבצים:
  - a. עשרים ואחד קבצים למימוש המשחק "מלך הפלאפל".
  - b. שלושה קבצי בדיקה לשלוש מחלקות שונות.
- על שמות הקבצים להיות **זהים** להגדרה בתרגיל. **מלבד מחלקות החריגות, עליכם לממש כל מחלקה בקובץ נפרד, כאשר שם הקובץ מתאים לשם המחלקה.**
2. עליכם לשמור ב-PyCharm את כל הקבצים באותה תיקייה על מנת שתוכלו לייבא (import) מחלקה ללא נתיב מלא לקובץ. להזכירכם, על מנת לייבא מחלקה **ClassX** מקובץ **FileName.py** עליכם להשתמש בסינטקס הבא:

```
from FileName import ClassX
```

לפניכם רשימת הקבצים שעליכם להגיש:

Angry.py  
ArrivalTimeServingStrategy.py  
Calm.py  
Chill.py  
Customer.py  
Dish.py  
exceptions.py  
Explosive.py  
FalafelStall.py  
FixedOrdersStrategy.py  
Furious.py  
Game.py  
LeastPatienceCustomerServingStrategy.py  
LongestWaitingTimeServingStrategy.py  
Mood.py  
OrdersStrategy.py  
Personality.py  
RandomOrdersStrategy.py  
ServingStrategy.py  
test\_Calm.py  
test\_Dish.py  
test\_TypeA.py  
TypeA.py  
TypeB.py

## חלק א'

בחלק זה יפורטו המחלקות אשר עליכם לממש.

## מחלקות אבסטרקטיות

עליכם ליצור ארבע מחלקות אבסטרקטיות: OrdersStrategy, ServingStrategy, Personality ו-Mood. עבור כל מחלקה, עליכם להחליט:

- אילו נתונים תחזיק כל מחלקה
- אילו שיטות צריכות להיות בא
- אילו מהשיטות שלה הן אבסטרקטיות ואילו מהן צריכות להתממש במחלקה

## המחלקה Dish

המחלקה Dish מייצגת מנה בדוכן פלאפל ומיוצגת על ידי שדה יחיד:

**ingredients** - רשימה של מחרוזות המחזיקה את הרכיבים המרכיבים את המנה. אין חשיבות לסדר הרכיבים ורכיב יכול להופיע יותר מפעם אחת.

ממשו את מחלקת Dish:

```
def __init__(self, ingredients=None):
```

ממשו בנאי המאתחל את שדה המחלקה. ניתן להניח כי אם תסופק, ingredients היא רשימה.

```
def add_ingredient(self, ingredient):
```

השיטה מוסיפה רכיב לרשימת המרכיבים.

ניתן להניח כי ingredient מטיפוס str.

```
def get_ingredients(self):
```

השיטה מחזירה את רשימת המרכיבים.

```
def __eq__(self, other):
```

השיטה מגדירה את אופן ההשוואה בין שתי מנות.

שתי מנות נחשבות שוות אם:

- שתיהן מטיפוס Dish.
- יש להן את אותו מספר רכיבים.
- הרכיבים שלהן זהים (אין חשיבות לסדר של הרכיבים במנה).

```
def __repr__(self):
```

השיטה מחזירה ייצוג מחרוזתי של המנה, בצורה של רשימת רכיבים עטופה בכוכביות. בפורמת הבא (שימו לב לרווחים):

```
* <list_of_ingredients> *
```

לדוגמה: אם המנה מכילה חומס וצ'יפס, יוחזר:

```
* humus, french fries *
```

אין חשיבות לסדר ההדפסה.

## המחלקה Customer

מחלקה זו מייצגת לקוח של דוכן הפלאפל ומיוצגת לפי השדות הבאים:

**name** – משתנה מטיפוס מחרוזת המייצג את שם הלקוח.  
**mood** – משתנה מטיפוס Mood, המייצג את מצב רוחו של הלקוח ומשפיע על סבלנותו.  
**personality** – משתנה מטיפוס Personality, המייצג את אישיותו של הלקוח המשפיעה על מצב רוחו.  
**patience** – משתנה מטיפוס float, המייצג את מידת הסבלנות הנוכחית של הלקוח.  
**initial\_patience** – משתנה מטיפוס float, המייצג את מידת הסבלנות של הלקוח, ערך ברירת המחדל הוא 100.  
**arrive\_time** – משתנה מטיפוס int, המייצג את הזמן שבו הלקוח הגיע לדוכן בשניות. ערך זה מאותחל ע"י שימוש בחבילה time והוא מהווה את הזמן הנוכחי בעת יצירת המופע. עם זאת, ניתן גם להגדיר ערך מותאם אישית עבור משתנה זה, כדי לאפשר בדיקות של הקוד עם זמני הגעה שונים.

כדי לייצר את **arrive\_time** עליכם להשתמש בחבילה time לאחר ייבוא בצורה הבאה:  
`arrive_time=int(time.time())`

### ניתן להניח כי הקלט של השדות תקין.

ממשו את מחלקת Customer:

```
def __init__(self, name, mood, personality, initial_patience=100, arrive_time=None)
```

בנאי המאתחל את שדות המחלקה הרלוונטים בהתאם לארגומנטים name, mood, personality, initial\_patience. בהעדר הארגומנט initial\_patience עליכם לאתחל אותו עם ערך ברירת המחדל 100.

```
def get_mood(self)
```

השיטה מחזירה את מצב הרוח הנוכחי של הלקוח.

```
def get_waiting_time(self, current_time=None)
```

השיטה מחשבת ומחזירה את זמן ההמתנה של הלקוח בדוכן (בשניות), כלומר כמה זמן עבר מאז שנוצרה ההזמנה (הזמן הנוכחי פחות זמן ההגעה). אם הזמן הנוכחי לא מסופק יש לאתחל אותו ב `int(time.time())`.  
**ניתן להניח כי `current_time` מטיפוס float.**

```
def get_patience(self)
```

השיטה מחזירה את מידת הסבלנות הנוכחית של הלקוח מעוגלת ל 2 ספרות אחרי הנקודה.

```
def update(self, waiting_time=None):
```

השיטה נועדה לעדכן את המצב הדינמי של הלקוח במשחק. בשלב הראשון, היא מעדכנת את ערך הסבלנות של הלקוח על פי מצב הרוח הנוכחי שלו. לאחר מכן, השיטה מעדכנת את מצב הרוח של הלקוח בהתבסס על האישיות שלו. אם זמן ההמתנה לא מסופק יש לאתחל אותו כזמן **ההמתנה** של הלקוח.  
**ניתן להניח כי `waiting_time` מטיפוס float.**

```
def __repr__(self)
```

מחזירה מחרוזת המייצגת את הלקוח בתוך מסגרת. התכונות המוצגות הן שם, מצב רוח, אישיות וסבלנות.

כדי לייצר את המסגרת:

- יש לחשב את אורך השורה הארוכה ביותר (כולל שם התכונה והערך שלה).
- מוסיפים 4 לאורך השורה הארוכה ביותר כדי להתאים את הרווחים ואת הגבולות:
- 2 רווחים משני צדי הטקסט בתוך התיבה.
- 2 תווים גבוליים (\*).

לדוגמה:

```
*****
* name: 1                      *
* mood: Chill                  *
* personality: TypeA           *
* patience: 5.0                *
*****
```

## המחלקה TypeA

המחלקה TypeA מייצגת אישיות. למחלקה TypeA אין שדות.

ממשו את מחלקת TypeA:

```
def adjust_mood(self, mood, waiting_time)
```

שיטה זו מחזירה את מצב הרוח החדש של הלקוח בהתבסס על מצב הרוח הנוכחי שלו וזמן ההמתנה בדוכן. אם הזמן שהלקוח חיכה בדוכן גדול מ-40 שניות, השיטה תחזיר מצב הרוח Explosive. אם הלקוח חיכה מעל 30 שניות ומצב רוחו הנוכחי הוא לא Explosive, השיטה תחזיר מצב רוח Furious. אם הלקוח חיכה מעל 20 שניות ומצב רוחו הנוכחי הוא לא Explosive או Furious, השיטה תחזיר מצב רוח Angry. ניתן להניח כי mood מטיפוס Mood ו waiting\_time מטיפוס float.

```
def __repr__(self)
```

מחזירה את המילה "TypeA", שמייצגת את סוג האישיות של הלקוח.



## המחלקה TypeB

המחלקה TypeB מייצגת אישיות. למחלקה TypeB אין שדות.

ממשו את מחלקת TypeB:

```
def adjust_mood(self, mood, waiting_time)
```

שיטה זו מחזירה את מצב הרוח החדש של הלקוח בהתבסס על מצב הרוח הנוכחי שלו וזמן ההמתנה בדוכן. אם הזמן שהלקוח חיכה בדוכן גדול מ-120 שניות ומצב רוחו הנוכחי הוא Furious, השיטה תחזיר מצב רוח Explosive. אם הזמן שהלקוח חיכה בדוכן גדול מ-90 שניות ומצב רוחו הנוכחי הוא Angry, השיטה תחזיר מצב רוח Furious. אם הזמן שהלקוח חיכה בדוכן גדול מ-60 שניות ומצב רוחו הנוכחי הוא Calm, השיטה תחזיר מצב רוח Angry. אם הזמן שהלקוח חיכה בדוכן גדול מ-40 שניות ומצב רוחו הנוכחי הוא Chill, השיטה תחזיר מצב רוח Calm. ניתן להניח כי mood מטיפוס Mood ו waiting\_time מטיפוס float.

```
def __repr__(self)
```

מחזירה את המילה "TypeB", שמייצגת את סוג האישיות של הלקוח.

## המחלקה Angry

המחלקה Angry מייצגת מצב רוח כועס ומיוצגת על ידי השדה הבא:

**strength** – משתנה מטיפוס int, מספר טבעי גדול מ-0 המייצג את עוצמת השפעת מצב הרוח על סבלנות הלקוח.

ממשו את מחלקת Angry:

```
def __init__(self, strength=2):
```

בנאי המקבל את הארגומנט strength ומאתחלת את שדות המחלקה. בהעדר הארגומנט המתאים יש לאתחל את השדה strength עם ערך ברירת המחדל 2. ניתן להניח כי strength מטיפוס int.

```
def get_patience_factor(self, waiting_time)
```

השיטה מקבלת כקלט waiting\_time, משתנה מטיפוס float, ומחזירה את מידת ההשפעה של מצב הרוח על הסבלנות ע"י החישוב הבא:

$$Factor = round(1.3^{\frac{<waiting\_time>}{5}} * <strength>, 2)$$

ניתן להניח כי waiting\_time מטיפוס float.

```
def __repr__(self):
```

מחזירה את המילה "Angry", שמייצגת את מצב הרוח של הלקוח.

## המחלקה Furious

המחלקה Furious מייצגת מצב רוח זועם ומיוצגת על ידי השדה הבא:

**strength** – משתנה מטיפוס int, מספר טבעי גדול מ-0 המייצג את עוצמת השפעת מצב הרוח על סבלנות הלקוח.

ממשו את מחלקת Furious:

```
def __init__(self, strength=2):
```

בנאי המקבל את הארגומנט strength ומאתחלת את שדות המחלקה. בהעדר הארגומנט המתאים יש לאתחל את השדה strength עם ערך ברירת המחדל 2. ניתן להניח כי strength מטיפוס int.

```
def get_patience_factor(self, waiting_time)
```

השיטה מקבלת כקלט waiting\_time, משתנה מטיפוס float. ומחזירה את מידת ההשפעה של מצב הרוח על הסבלנות ע"י החישוב הבא:

יש להחזיר את התוצאה כפולה פי 2 מהתוצאה המתקבלת מהמחלקה Angry.

ניתן להניח כי waiting\_time מטיפוס float.

```
def __repr__(self):
```

מחזירה את המילה "Furious", שמייצגת את מצב הרוח של הלקוח.

## המחלקה Explosive

המחלקה Explosive מייצגת מצב רוח נפיץ ומיוצגת על ידי השדה הבא:

**strength** – משתנה מטיפוס int, מספר טבעי גדול מ-0 המייצג את עוצמת השפעת מצב הרוח על סבלנות הלקוח.

ממשו את מחלקת Explosive:

```
def __init__(self, strength=2):
```

בנאי המקבל את הארגומנט strength ומאתחלת את שדות המחלקה. בהעדר הארגומנט המתאים יש לאתחל את השדה strength עם ערך ברירת המחדל 2. ניתן להניח כי strength מטיפוס int.

```
def get_patience_factor(self, waiting_time)
```

השיטה מקבלת כקלט waiting\_time, משתנה מטיפוס float. ומחזירה את מידת ההשפעה של מצב הרוח על הסבלנות ע"י החישוב הבא:

$$Factor = round(1.3^{\frac{<waiting\_time> * <strength>}{5}}, 2)$$

ניתן להניח כי waiting\_time מטיפוס float.

```
def __repr__(self):
```

מחזירה את המילה "Explosive", שמייצגת את מצב הרוח של הלקוח.

## המחלקה Calm

המחלקה Calm מייצגת מצב רוח רגוע ומיוצגת על ידי השדה הבא:

**strength** – משתנה מטיפוס int, מספר טבעי גדול מ-0 המייצג את עוצמת השפעת מצב הרוח על סבלנות הלקוח.

ממשו את מחלקת Calm:

```
def __init__(self, strength=2):
```

בנאי המקבל את הארגומנט strength ומאתחלת את שדות המחלקה. בהעדר הארגומנט המתאים יש לאתחל את השדה strength עם ערך ברירת המחדל 2. ניתן להניח כי strength מטיפוס int.

```
def get_patience_factor(self, waiting_time)
```

השיטה מקבלת כקלט waiting\_time, משתנה מטיפוס float, ומחזירה את מידת ההשפעה של מצב הרוח על הסבלנות ע"י החישוב הבא:

$$Factor = round(1.05^{\frac{<waiting\_time>}{5}} * <strength>, 2)$$

ניתן להניח כי waiting\_time מטיפוס float.

```
def __repr__(self):
```

מחזירה את המילה "Calm", שמייצגת את מצב הרוח של הלקוח.

## המחלקה Chill

המחלקה Chill מייצגת מצב רוח צונן ומיוצגת על ידי השדות הבאים:

**strength** – משתנה מטיפוס int, המייצג את עוצמת השפעת מצב הרוח על סבלנות הלקוח.  
**chill\_modifier** – משתנה מטיפוס float, בין 1 ל-0, המייצג את מידת הרוגע של מצב הרוח.

ממשו את מחלקת Chill:

```
def __init__(self, strength=2, chill_modifier=0.5):
```

בנאי המקבל את הארגומנטים strength ו-chill\_modifier, ומאתחלת את שדות המחלקה. בהעדר הארגומנטים המתאימים יש לאתחל את השדות strength ו-chill\_modifier עם ערכי ברירת המחדל 2 ו-0.5 בהתאמה. ניתן להניח כי strength מטיפוס int ו-chill\_modifier מטיפוס float.

```
def get_patience_factor(self, waiting_time)
```

השיטה מקבלת כקלט waiting\_time, משתנה מטיפוס float, ומחזירה את מידת ההשפעה של מצב הרוח על הסבלנות ע"י החישוב הבא:

$$Factor = round(1.05^{\frac{<waiting\_time>}{5}} * <strength> * <chill\_modifier>, 2)$$

ניתן להניח כי waiting\_time מטיפוס float.

```
def __repr__(self):
```

מחזירה את המילה "Chill", שמייצגת את מצב הרוח של הלקוח.

## המחלקה ArrivalTimeServingStrategy

המחלקה מייצגת אסטרטגיה של תעדוף הזמנות לפי זמן ההגעה שלהן.

ממשו את מחלקת ArrivalTimeServingStrategy:

*def select\_next\_order(self, orders):*

השיטה מקבלת את המשתנה **orders**, מילון המחזיק את ההזמנות שבוצעו. המפתח הוא מזהה ההזמנה (מטיפוס int, מספר טבעי הגדול מ-0), והערך הוא זוג של לקוח ומנה (טאפל המכיל מופע של המחלקה Customer ומופע של המחלקה Dish בסדר המצוין). ומחזירה את **מזהה** ההזמנה הבאה על פי זמן ההגעה שלה, במידה וכמה הזמנות הגיעו באותו הזמן תיבחר ההזמנה שנכנסה קודם למילון. במידה ולא קיימות הזמנות ב **orders** תזרק שגיאה `OrderOutOfBoundsException`. **ניתן להניח כי orders הוא קלט תקין כפי שתואר.**

## המחלקה LongestWaitingTimeServingStrategy

המחלקה מייצגת אסטרטגיה של תעדוף הזמנות לפי זמן ההמתנה שלהן.

ממשו את מחלקת LongestWaitingTimeServingStrategy:

*def select\_next\_order(self, orders):*

השיטה מקבלת את המשתנה **orders**, מילון המחזיק את ההזמנות שבוצעו. המפתח הוא מזהה ההזמנה (מטיפוס int, מספר טבעי הגדול מ-0), והערך הוא זוג של לקוח ומנה (טאפל המכיל מופע של המחלקה Customer ומופע של המחלקה Dish בסדר המצוין). ומחזירה את **מזהה** ההזמנה שזמן ההמתנה שלה הוא הארוך ביותר. במידה וכמה הזמנות מחכות זמן זהה תיבחר ההזמנה שנכנסה קודם למילון. במידה ולא קיימות הזמנות ב **orders** תזרק שגיאה `OrderOutOfBoundsException`. **ניתן להניח כי orders הוא קלט תקין כפי שתואר.**

## המחלקה LeastPatienceCustomerServingStrategy

המחלקה מייצגת אסטרטגיה של תעדוף הזמנות לפי סבלנותו של הלקוח:

ממשו את מחלקת LeastPatienceCustomerServingStrategy:

*def select\_next\_order(self, orders):*

השיטה מקבלת את המשתנה **orders**, מילון המחזיק את ההזמנות שבוצעו. המפתח הוא מזהה ההזמנה (מטיפוס int, מספר טבעי הגדול מ-0), והערך הוא זוג של לקוח ומנה (טאפל המכיל מופע של המחלקה Customer ומופע של המחלקה Dish בסדר המצוין). השיטה מחזירה את **מזהה** ההזמנה שהסבלנות של הלקוח שלה היא הנמוכה ביותר. במידה וכמה הזמנות עולות אותו הדבר תיבחר ההזמנה שנכנסה קודם למילון.

במידה ולא קיימות הזמנות ב **orders** תזרק שגיאה OrderOutOfBoundsException.

ניתן להניח כי orders הוא קלט תקין כפי שתואר.

## המחלקה FalafelStall

מחלקה זו מייצגת דוכן פלאפל שמנהל הזמנות של לקוחות, משרת מנות, ומנהל את המלאי והכספים שלו. המחלקה כוללת רשימה של רכיבים זמינים, מונה הזמנות סטטי, ושיטות שונות לביצוע פעולות כמו הוספת הזמנה, קבלת פרטי הזמנה, וסילוק הזמנות.

המחלקה מתוארת ע"י השדות הבאים:

**orders** - מילון שמחזיק את ההזמנות שבוצעו. כל מפתח ב-orders הוא מזהה ההזמנה (מספר טבעי הגדול מ-0), והוא ממופה לטאפל המכיל לקוח (מופע של המחלקה Customer) ומנה (מופע של המחלקה Dish) בסדר המצוין.

**strategy** - אסטרטגיה לבחירת ההזמנה הבאה שתוגש. (למשל, לפי סדר ראשון-נכנס-ראשון-יוצא).

**ingredient\_prices** – מילון מחירים של רכיבים הקיימים בדוכן הפלאפל, כאשר כל מפתח הוא רכיב (מחרוזת) הממופה למחירו (float או int).

**money** - משתנה מטיפוס float שמייצג את הסכום המצטבר שהדוכן הרוויח.

**order\_counter** – מחזיק את כמות ההזמנות שבוצעו עד כה בדוכן.

ממשו את המחלקה FalafelStall

*def \_\_init\_\_(self, strategy, ingredient\_prices):*

בנאי המאתחל את שדות המחלקה strategy, ingredient\_prices בהתאם לקלט.

ניתן להניח כי הקלט של השדות תקין.

*def order(self, customer, dish):*

השיטה מקבלת בקלט את customer (מופע של המחלקה Customer) ומנה (מופע של המחלקה Dish). מוסיפה הזמנה חדשה למערכת. כל הזמנה חדשה מקבלת מזהה אוטומטי השווה לכמות ההזמנות שהוזמנו עד כה  $order\_counter + 1$ . במידה וקיימים רכיבים במנה שאינם קיימים בדוכן הפלאפל תיזרק שגיאה `NoSuchIngredientException`. על השיטה להחזיר את מזהה ההזמנה שנוצרה. ניתן להניח כי הפרמטר dish מטיפוס Dish וכי customer מטיפוס Customer.

*def get\_next\_order\_id(self):*

השיטה מחזירה את מזהה ההזמנה הבאה לפי האסטרטגיה שהוגדרה. אם אין הזמנות, זורקת חריגת `OrderOutOfBoundsException`.

*def serve\_dish(self, order\_id, dish):*

השיטה מגישה מנה לפי מזהה ההזמנה. השיטה מקבלת בקלט את מזהה את הזמנה (order\_id) משתנה מטיפוס (int) ואת המנה הצפויה (מופע של המחלקה Dish) ובודקת אם המנה שהוגשה תואמת את המנה שהוזמנה. בהתאם לתוצאת הבדיקה השיטה מעדכנת את הרווחים של הדוכן (כלומר מעדכנת את השדה money). אם ההזמנה לא קיימת או לא תואמת, נזרקות חריגות מתאימות. לא ניתן להניח תקינות הקלט.

*def remove\_order(self, order\_id):*

השיטה מקבלת מזהה של הזמנה (מטיפוס int) ומסירה את הזמנה מהמילון ההזמנות לפי מזהה ההזמנה. אם המזהה לא קיים במערכת, נזרקת חריגת `NoSuchOrderException`. לא ניתן להניח תקינות הקלט.

*def get\_order(self, order\_id):*

השיטה מקבלת מזהה של הזמנה (מטיפוס int) ומחזירה את ההזמנה לפי המזהה. זורקת חריגה `NoSuchOrderException` אם ההזמנה לא קיימת במערכת. לא ניתן להניח תקינות הקלט.

*def calculate\_cost(self, dish):*

השיטה מקבלת בקלט את מנה (משתנה מטיפוס Dish) ועליה להחזיר את המחיר הכולל של המנה. אם אחד הרכיבים במנה לא נמצא ברשימת הרכיבים הזמינים של הדוכן, נזרקת חריגת `NoSuchIngredientException`. ניתן להניח כי dish מטיפוס Dish.

*def get\_orders(self):*

השיטה מחזירה את מילון ההזמנות.

*def get\_earning(self):*

השיטה מחזירה את הסכום המצטבר שהדוכן הרוויח.

## המחלקה RandomOrdersStrategy

המחלקה RandomOrdersStrategy מייצגת אסטרטגיית יצירת הזמנות רנדומאלית. מופע של המחלקה מיוצג ע"י השדות הבאים:

**current** – שדה שתפקידו לעקוב אחר **כמות ההזמנות שיוצרו**. משתנה מטיפוס int.  
**n\_orders** – שדה שתפקידו להציב מקסימום של הזמנות שיווצרו ע"י האיטרטור. ערך שלילי מציין שהאיטרטור יימשך ללא הגבלה (ללא סוף). משתנה מטיפוס int.  
**max\_dishes** – שדה המייצג מספר המנות המקסימלי להפקת בכל צעד (איטרציה), משתנה מטיפוס int מספר טבעי הגודל מ-0.  
**ingredients** – רשימה של מרכיבים, ממנה יבחרו מרכיבים למנה.  
**max\_ingredients** – שדה המתאר את מספר הרכיבים המקסימלי שיכולים להיות במנה. משתנה מטיפוס int מספר טבעי הגודל מ-0.

ממשו את מחלקת RandomOrdersStrategy:

```
def __init__(self, max_dishes, max_ingredients, ingredients, n_orders=-1):
```

בנאי המאתחל את שדות המחלקה בהתאם לארגומנטים שהבנאי מקבל. בהעדר ארגומנט n\_orders יש לאתחול ב-1.

ניתן להניח כי הקלט של השדות תקין.

```
def __iter__(self):
```

השיטה מחזירה איטרטור על המחלקה.

```
def __next__(self):
```

השיטה מחזירה רשימה של הזמנות **orders** כך שכל הזמנה היא טאפל של לקוח (Customer) ומנה (Dish) בהתאמה. כמות המנות שתייצר השיטה הוא מספר שלם בין 0 ל-**max\_dishes** וכמות הרכיבים בכל מנה תהיה מספר רנדומאלי בין 1 ל-**max\_ingredients**.  
מצב הרוח והאישיות של הלקוח נבחרים באופן רנדומלי בין כל מצבי הרוח **והאישיות** עם הערכים הדיפולים שלהם.

שמו של הלקוח הוא שם ייחודי, ונקבע **לפי כמות ההזמנות שיווצרו עד כה ועוד 1** (השתמשו ב-**current+1** כדי לתת שם ללקוח).

**current** עוקב אחר כמות ההזמנות שיוצרה בצעד הנוכחי.

במחלקה זאת תצטרכו להשתמש בספריית random על מנת לקבל ערכים רנדומאליים. שימו לב ל random לפונקציות random.choice, random.random ו random.chocies.

## המחלקה FixedOrdersStrategy

המחלקה FixedOrdersStrategy מייצגת אסטרטגיית יצירת מנות קבועה, מתוארת על ידי השדות הבאים:

**lst\_orders** – שדה מטיפוס רשימה מקוננת של הזמנות, כל איבר ברשימה מכיל רשימה של הזמנות.  
**index** – שדה שתפקידו לעקוב אחר ההתקדמות הנוכחית של האיטרטור. משתנה מטיפוס int מספר טבעי הגדול מ-0.

ממשו את מחלקת FixedOrdersStrategy:

```
def __init__(self, lst_orders):
```

בנאי המאתחל את שדות המחלקה.

ניתן להניח כי lst\_orders הוא קלט תקין כפי שתואר.

```
def __iter__(self):
```

השיטה מחזירה איטרטור על המחלקה.

```
def __next__(self):
```

השיטה מחזירה רשימה של הזמנות מ **lst\_orders** לפי ה **index** הנוכחי שלה.

## המחלקה Game

מחלקת Game היא המוח של המשחק ותפקידה לנהל יצירה של דוכן פלאפל, יצירה של הזמנות עבור הדוכן, עדכון מצב הרוח והסבלנות של הלקוחות וכמובן, הכנתן של ההזמנות ע"י השחקן, **כפי שתואר בהקדמה.**

המחלקה מתוארת על ידי השדות הבאים:

**orders\_strategy** – משתנה המייצג את האסטרטגיה להפקת הזמנות (למשל, אקראית או קבועה).  
**serving\_strategy** – משתנה המייצג את האסטרטגיה להגשת הזמנות (למשל, לפי זמן הגעה או זמן המתנה).

**ingredient\_prices** – מילון מחירים של רכיבים הקיימים בדוכן הפלאפל, כאשר כל מפתח הוא רכיב (מחרוזת) הממופה למחירו (float או int).

**game\_start** – זמן התחלת המשחק (בעזרת `int(time.time())`).

**lives** – מספר הניסיונות של השחקן (התחלה ב-3).

**ingredient\_dictionary** – מילון שמכיל את הרכיבים הזמינים במערכת **(ראו דוגמה מטה)**.



ממשו את מחלקת Game:

`def __init__(self, orders_strategy, serving_strategy, ingredient_prices):`

בנאי המאתחל את שדות המחלקה.

ניתן להניח כי הקלט הוא קלט תקין כפי שתואר.

`def get_lives(self):`

השיטה מחזירה את מספר הניסיונות של השחקן.

`def get_game_duration(self, current_time=None):`

השיטה מחשבת ומחזירה את זמן המשחק (בשניות), כלומר כמה זמן עבר מאז שהתחיל המשחק (הזמן הנוכחי פחות זמן ההתחלה). אם הזמן הנוכחי לא מסופק יש לאתחל אותו ב `int(time.time())`.

ניתן להניח כי `current time` מטיפוס `float`.

`def run(self):`

השיטה תאתחל אובייקט FalafelStall ותכין הזמנות כל עוד למשתמש יש עוד ניסיונות (`lives`) בעזרת אסטרטגיית יצירת ההזמנות (בשדה `orders_strategy`) השיטה תבצע הזמנות לדוכן הפלאפל (יכול להיות כי יוחזרו רשימות ריקות מאסטרטגיית יצירת ההזמנות, וזה בסדר, יש להמשיך בתכנית ולחכות עד שגיעו הזמנות לדוכן).

בשלב הזה, הדוכן מוציא את ההזמנות אחת אחרי השניה לפי אסטרטגיית ההגשות (השדה `serving_strategy`) ומציג לשחקן את ההזמנה שעליו להכין בצורה הבאה:

Customer:

{customer}

Dish: {ordered\_dish}

לדוגמה:

Customer:

```
*****
* name: 1 *
* mood: Chill *
* personality: TypeA *
* patience: 5.0 *
*****
```

Dish: \* falafel, humus, french fries, falafel \*

לאחר מכן, יוצג לשחקן תפריט הרכיבים (`ingredient_dictionary`):

```
Insert ingredients:
0: green salad
1: falafel
2: french fries
3: coleslaw
4: fried eggplants
5: tachina
6: humus
```

המערכת מצפה שהשחקן יכניס את הרכיבים שיש למנה מתוך מילון הרכיבים, ייצור את המנה (Dish) ויגיש אותה ללקוח.

כדי שהשחקן ייצור את המנה עליו להכניס רצף של מספרים, המופרדים ברווח, המייצגים את המרכיבים של המנה (בסדר כלשהו). לדוגמה, עבור המנה למעלה על השחקן לספק כקלט את המספרים:

```
1 6 4 6
```

במידה והוכנס מספר שלא קיים בתפריט (לדוגמה 9) המרכיב ייחשב כמחוזת ריקה ("").

לאחר הזנת המנה הדוכן מגיש את ההזמנה ללקוח, אם ההגשה הייתה מוצלחת ההזמנה תוסר ממילון ההזמנות של דוכן הפלאפל.

שגיאות עלולות להתרחש במהלך יצירת ההזמנה:

במידה ונוצרה מנה עם רכיב שלא קיים יש להדפיס את הטקסט מטה ולהמשיך בתכנית:

```
Failed to create a Dish
{e}
please retry.
```

במידה ונוצרה מנה שלא תואמת את ההזמנה של הלקוח יש להדפיס ולהמשיך בתכנית:

```
Failed to serve a Dish to customer
{e}
```

במידה ולא יהיו הזמנות המוכות להכנה, יש לחכות עד שתיכנס הזמנה לדוכן.

בשלב האחרון על השיטה לעדכן את מצב הרוח של הלקוחות שההזמנות שלהן עדיין לא הוגשו. במידה והסבלנות של אחד הלקוחות קטנה או שווה ל 0 יש להסיר את ההזמנה שלו מהדוכן ולהוריד את כמות הניסיונות של השחקן (**lives**) ב 1.

המשחק נגמר כאשר הניסיונות של השחקן קטנים או שווים ל 0 ובסיומו תודפס הודעת "Game Over" בצירוף הציון הנוכחי (הרווח של הדוכן) והמשחק נעצר:

```
Game Over
score: {score}
```

לצורך הרצת התרגיל סיפקנו לכם קובץ `main.py` אשר תוכלו להשתמש בו על מנת להריץ את המשחק לאחר שתיצרו את המחלקות.

## חריגות (Exceptions)

עליכם לממש מספר מחלקות מטיפוס Exception בקובץ בשם exceptions.py. במהלך מימוש המשחק יהיה עליכם לזרוק את החריגות המתאימות בהתאם לדרישות שיפורטו במחלקות האחרות.

### המחלקה NoSuchIngredientException

החריגה תיזרק כאשר מנסים להרכיב מנה מאחד הרכיבים אשר אינו נמצא בדוכן הפלאפל. בחריגה זו עליכם לדרוס את השיטה `__str__` של המחלקה `NoSuchIngredientException` כך שתחזיר מחרוזת המייצגת את החריגה לפי הפורמט הבא:

```
Error:\n"{ingredient}" is an invalid ingredient.
```

דוגמא:

```
try:
    raise NoSuchIngredientException ("banana")
except Exception as e:
    print(e)
```

יודפס:

```
Error:
"banana" is an invalid ingredient.
```

### המחלקה NotCustomerDishException

החריגה תיזרק כאשר מנסים להגיש ללקוח מנה שלא תואמת את ההזמנה.

בחריגה זו עליכם לדרוס את השיטה `__str__` של המחלקה `NotCustomerDishException` כך שתחזיר מחרוזת המייצגת את החריגה לפי הפורמט הבא:

```
Error:\nThe suggested dish:\t{suggested_dish}\nis not as
expected:\t{expected_dish}.
```

דוגמא:

```
try:
    raise NotCustomerDishException(Dish(['a']), Dish(['a',
    'b']))
except Exception as e:
    print(e)
```

יודפס:

```
Error:
The suggested dish: * a *
is not as expected: * a, b *
```

## המחלקה NoSuchOrderException

החריגה תיזרק כאשר מנסים לגשת להזמנה שלא קיימת.  
 בחריגה זו עליכם לדרוס את השיטה `__str__` של המחלקה `NoSuchOrderException` כך שתחזיר מחרוזת המייצגת את החריגה לפי הפורמט הבא:

```
Error:\nOrderID: "{order_id}" does not exist.
```

דוגמא:

```
try:
    raise NotCustomerDishException(1)
except Exception as e:
    print(e)
```

יודפס:

```
Error:
OrderID: "1" does not exist.
```

## המחלקה OrderOutOfBoundsException

החריגה תיזרק כאשר מנסים לגשת לרשימת הזמנות ריקה. אין צורך לבצע שינויים נוספים בחריגה זו.

## חלק ב' - Unit test

זאת הפעם הראשונה שהינכם מתמודדים עם מספר רב של קבצים, שיטות ומחלקות. על מנת שתוכלו לבדוק את התרגיל שלכם בצורה קלה ונוחה תשתמשו בכלי בדיקות תכנה שנקרא **unit test**. כל זה הוצג בתרגול מספר 7 ובמעדה מספר 7, וישמש אתכם בהמשך דרככם המקצועית.

עליכם לממש `UnitTest` לשלושת המחלקות `Calm`, `TypeA` ו-`Dish`, כאשר לכל אחת מהמחלקות יש ליצור לפחות 3 טסטים שונים.

כדי לראות כיצד מכינים `Unit Test` ניתן גם להסתכל באתר באתר של `jetbrains`:

<https://www.jetbrains.com/help/pycharm/testing-your-first-python-application.html#run-test-automatically>

לנוחיותכם סיפקנו גם קובץ `test_Game.py` אשר יאפשר להריץ `test` על המשחק ע"י הכנסת קלט אוטומטית.