

## תרגיל 5: תכונות מונחה עצמים

תאריך פרסום: 27.12.2024

תאריך הגשה: 23:59 ב-12.01.2025

מתרגל אחראי: טל מלול

משקל תרגיל: 5 נקודות

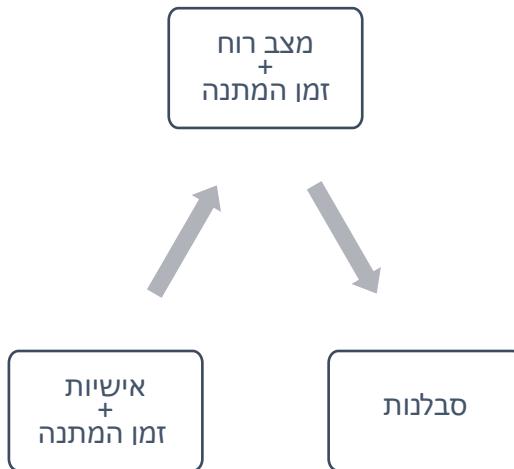
### dagim:

1. מומלץ לקרוא את כל התרגיל **לפni** המימוש. אנו ממליצים להבין היטב את ארכיטקטורת המערכת ולתכנן אותה על דף נייר **לפni** תחילת כתיבת הקוד.
2. בניגוד לתרגילים קודמים, **בתרגיל זה אין להניח שהקלט תקין**uboar בל שיטה / פונקציה מוגדרות הדרישות לגבי הקלט. הנחות על קלטים מסוימים יוצינו באופן מפורש. ובמקרה והקלט לא תקין עליכם לזרוק חריגה מתאימה.
3. מכיוון שהעבודה עמוסה מבחינת במות הקוד שעלייכם למש וכדי להקל עליכם - איןכם נדרש **לכתוב docstring בעבודה זו.**
4. בתרגיל **לא מצוין** באופן מפורש איך עליכם למש את הפתרון. לדוגמה, לא יצוין מתי עליכם לדחוס שיטה, להשתמש בשיטת האב, להגדיר מחלוקת\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 (לדוגמה, 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 **לפni** שאתם מתחילה לכתוב את הקוד (זו הפעם השנייה של הסעיף זה ולא במקרה..).

## מלך הפלאף: הקדמה

במקרה זה תתבקשו למשוך גרסה מצומצמת של המשחק "מלך הפלאף". במשחק אתם מנהלים דוכן פלאף והמטרה שלכם היא להרוויח כמה שיותר כסף על ידי מכירת מנוקט פלאף המזמנות ע"י הלקוחות. לבסוף מידת סבלנות מסוימת שקבעה בקצב שימוש מצב רוחו וזמן ההמתנה למנקה שלו. לבסוף יש אישיות שמכתיבת איך מצב הרוח שלו משתנה בהתאם בזמן ההמתנה. אם ללקוח נגמרת הסבלנות, הוא עוזב את הדוכן. אם מספר הלקוחות שעזבו חורג מסף מסוים – המשחק מסתיים.



### חוקי המשחק:

בכל שלב, מגיעים למועדון חדש ומצמינים מנוקט פלאף. עליו להכין את המנה לפי הזמנה ולהגיע אותה ללקוח לפני שהסבלנות שלו נגמרה.

### ניהול המשחק:

קיים שני מנגנונים אפשריים לניהול הזמן:

1. אקרואית – הזמנות נוצרות באופן אקראי.
2. קבועה מראש – הזמנות נוצרות באופן ידוע מראש.

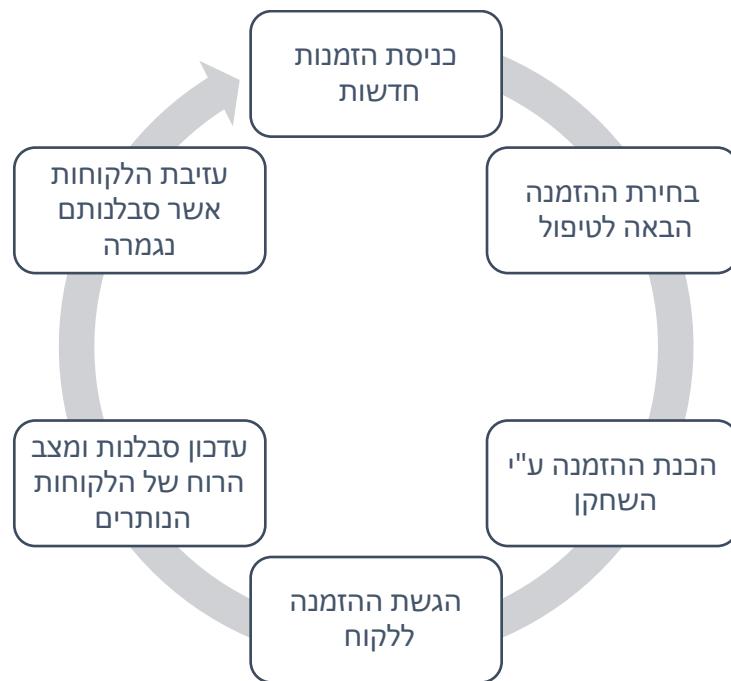
### בחירה ההזמנה הבאה לטיפול:

בכל נקודת זמן, עליו להחליט איזו הזמנה להכין בעתיד. לצורך כך, תמשכו מספר אסטרטגיות עברו החלטה מה תהיה ההזמנה הבאה לטיפול. למשל, אסטרטגיה אחת יכולה לטעוף הזמנות על פי סדר הגעה, אסטרטגיה שנייה על פי זמן המנתנה האחרון ביותר, ואסטרטגיה שלישית על פי סבלנותו של הלקוח.

### מהלך המשחק וסיומו:

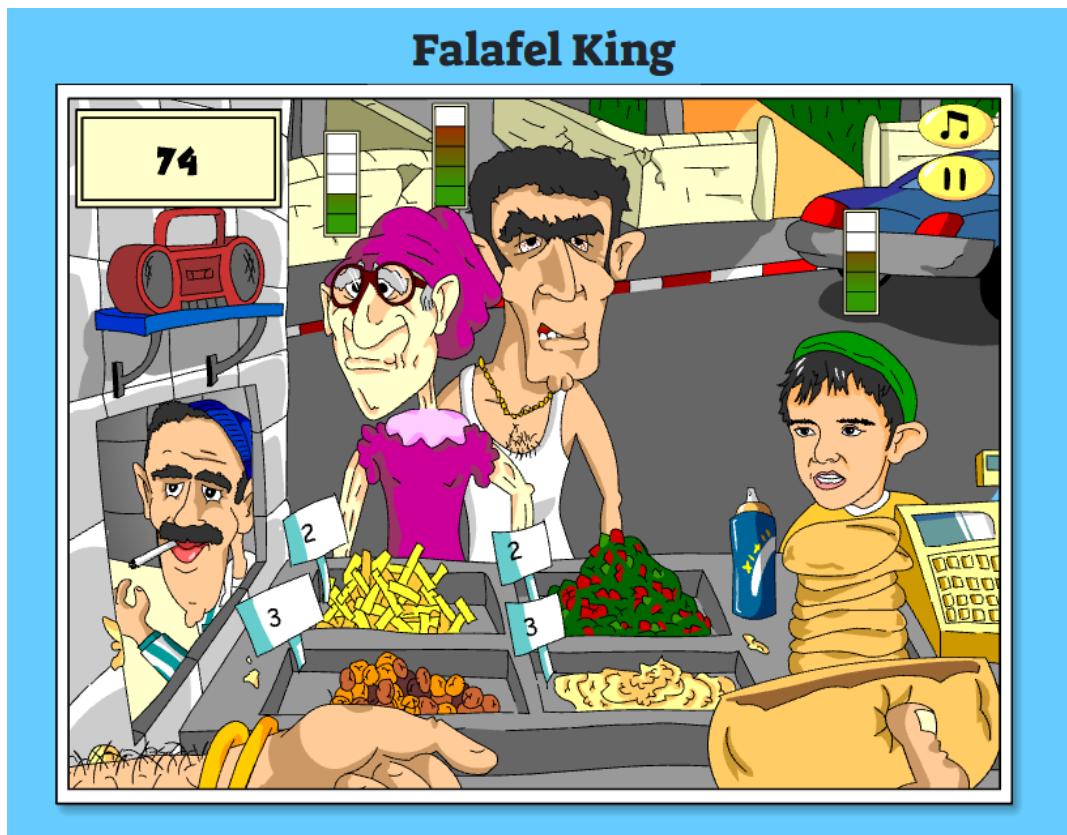
בכל שלב, לפי אחד מנגנוני ניהול הזמן, מגיעים למועדון החדש ומצמינים מנוקט פלאף. הזמן אליהם יתווסף להזמנות שכבר קיימות בדוכן הפלאף. לאחר מכן, לפי אחת מאסטרטגיות تعدוף הזמן, תיבחר הזמנה הבאה לטיפול. בעתיד, יוצג טפריט שיאפשר לכם להרכיב את הזמנה באמצעות הבנטת קלט, להציג ללקוח ולגבות תשלום עבור הזמנה. בכל סיבוב, יש לעדכן את סבלנותם ו מצב רוחם של לקוחות שהזמנתם עדין לא טופלה. אם סבלנותו של לקוח נגמרה, הוא יעדוב את הדוכן והזמןתו תוסר מרשימה הזמן. המשחק מסתיים כאשר כמות הלקוחות שעזבו (כגמירה להם הסבלנות) תעבור סף מסוים. בסיום, יוצג הסכום שהרוויחתם.

מצורפת תמונה המתארת את מהלך המשחק:



על מנת למש את המשחק תתבקשו לתקן ולמש מחלקות תוך יישום עקרונות תכונות מונחה עצמים (OOP) כפי שלמדתם בהרצאות ובתרגולים: אבסטרקציה – abstraction, הרכבה – encapsulation, ירושא – inheritance ורב-צורתיות – polymorphism –.

בצלחה!



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

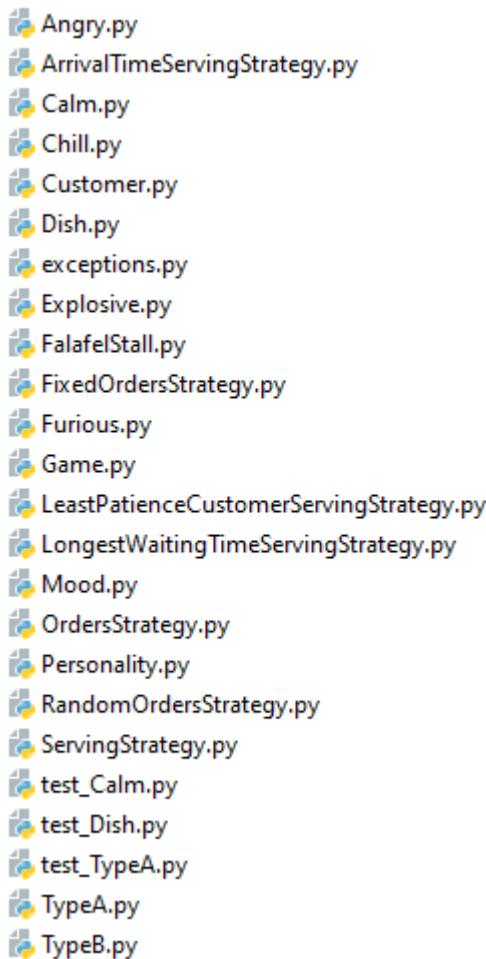
## מבנה ההגשה

### שימוש לב:

1. עליכם להגיש בעבודה זו 24 קבצים:
    - a. עשרים ואחד קבצים לימוש המשחק "מלך הפלפל".
    - b. שלושה קבצי בדיקה לשולש מחלקות שונות.
- על שמות הקבצים להיות זהים להגדלה בתרגיל. מלבד מחלקות החיריגות, עליכם למשתמש בכל מחלקה בקובץ נפרד, כאשר שם הקובץ מתאים לשם המחלקה.
2. עליכם לשמר ב-PyCharm את כל הקבצים באותה תיקייה על מנת שתוכלו לייבא (import) מחלקה ללא נטibus מלא לקובץ. להזיכרכם, על מנת לייבא מחלקה **ClassName** מקובץ **FileName.py** עליכם להשתמש בסינטקס הבא:

```
from FileName import ClassX
```

לפניכם רשימת הקבצים שעלייכם להגיש:



## חלק א'

בחלק זה יפורטו המחלקות אשר עליים למש.

### מחלקות אבסטרקטיות

עליכם ליצור ארבע מחלקות אבסטרקטיות: Mood-ו OrdersStrategy, ServingStrategy, Personality: עבורי כל מחלוקת, עליכם להחליט:

- אילו נתונים תחזיק כל מחלוקת
- אילו שיטות צrüכות להיות בא
- אילו מהשיטות שלה הן אבסטרקטיות וailו מהן צrüכות להתמש בחלוקת

## המחלקה Dish

המחלקה Dish מייצגת מנה בדוקן פלאפל ומוצגת על ידי שדה יחיד:

- רשיימה של מחרוזת המחזיקה את הרכיבים המרכיבים את המנה. אין חשיבות לסדר הרכיבים ורכיב יכול להופיע יותר מפעם אחת.

ממשו את מחלקה Dish:

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

ממשו בנייה המתחילה את שדה המחלקה. ניתן להניח כי אם תספק, ingredients היא רשיימה.

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

השיטה מוסיפה רכיב לרשימה של הרכיבים.

ניתן להניח כי ingredient מופיע מטיפוס str.

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

**השיטה מחזירה את רשימת הרכיבים.**

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

השיטה מגדרה את אופן ההשוואה בין שתי מנות.

שתי מנות נחשבות שוות אם:

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

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

השיטה מחזירה יציג מחרוזתי של המנה, בצורה של רשימת רכיבים עטופה בכוכיות. בפורמת הבא  
(שימוש לב לרווחים):

```
* <list_of_ingredients> *
```

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

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

אין חשיבות לסדר הדפסה.

## המחלקה Customer

מחלקה זו מייצגת לקוּח של דוכן הפלאָפֵל ומיצגת לפי השודות הבאים:

- name** – משתנה מטיפוס מחרוזת המייצג את שם הלקוּח.
- mood** - משתנה מטיפוס Mood, המייצג את מצב רוחו של הלקוּח ומשפיע על סבלנותו.
- משתנה מטיפוס Personality, המייצג את אישיותו של הלקוּח המשפיעה על מצב רוחו.
- משתנה מטיפוס float, המייצג את מידת הסובלנות הנוכחיית של הלקוּח.
- initial\_patience** – משתנה מטיפוס float, המייצג את מידת הסובלנות של הלקוּח, ערך ברירת המחדל הוא 100.
- arrive\_time** – משתנה מטיפוס int, המייצג את הזמן שבו הגיעו הגיעו לדוכן בשניות. ערך זה מואתחל ע"י שימוש בחבילה time והוא מראה 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 ליצור את **arrive\_time** עליו להשתמש בחבילה time לאחר יבוא בצורה הבאה:  
`arrive_time=int(time.time())`

ניתן להניח כי הקלט של השודות תקין.

משמעותו של מושג Customer

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

בנאי המאותל את שדות המחלקה הרלוונטיים בהתאם לארגומנטים name, mood, personality, initial\_patience. בהעדר הארגומנט initial\_patience עליים לאותל אותו עם ערך ברירת המחדל 100.

`def get_mood(self)`

השיטה מחזירה את מצב הרוח הנוכחי של הלקוּח.

`def get_waiting_time(self, current_time=None)`

השיטה מחשבת ומהזירה את זמן ההמתנה של הלקוּח בדוכן (בשניות), כלומר כמה זמן עבר מאז שנוצרה ההזינה (זמן הנוכחי פחות זמן הגעה). אם הזמן הנוכחי לא מסופק יש לאותל אותו ב(`(None)`).  
ניתן להניח כי current\_time מטיפוס float.

`def get_patience(self)`

השיטה מחזירה את מידת הסובלנות הנוכחיית של הלקוּח מעוגלת ל 2 ספרות אחרי הנקודה.

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

השיטה נועדה לעדכן את המצב הדינמי של הלקוּח במהלך המשחק. בשלב הראשון, היא מעדכנת את ערך הסובלנות של הלקוּח על פי מצב הרוח הנוכחי שלו. לאחר מכן, השיטה מעדכנת את מצב הרוח של הלקוּח בהתאם על האישיות שלו. אם זמן ההמתנה לא מסופק יש לאותל אותו כזמן **ההמתנה** של הלקוּח.  
ניתן להניח כי waiting\_time מטיפוס float.

`def __repr__(self)`

מחזירה מחרוזת המייצגת את הלוקה בתוך מסגרת. התוכנות המוצגות הן שם, מצב רוח, אישיות וסבלנות.

כדי ליצר את המסגרת:

- יש לחשב את אורך השורה הארכוה ביוטר (כולל שם התוכנה והערך שלה).
- מוסיפים 4 לאות השורה הארכוה ביוטר כדי להתאים את הרוחותים ואת הגבולות:
- 2 רוחות משני צדי הטקסט בתוך התביה.
- 2 תווים גבוליים (\*).

לדוגמה:

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

TypeA המחלקה

המחלקה TypeA מייצגת אישיות. למחלקה TypeA אין שדות.

ממשו את מחלקה TypeA:

`def adjust_mood(self, mood, waiting_time)`

שיטה זו מחזירה את מצב הרוח החדש של הלוקה בהתאם במצב הרוח הנוכחי שלו וזמן המתנה בדוקן. אם הזמן שהלוקה חיכה בדוקן גדול מ-40 שניות, השיטה תחזיר מצב הרוח Explosive. אם הלוקה חיכה מעל 30 שניות ומצב רוחו הנוכחי הוא לא Explosive, השיטה תחזיר מצב רוח Furious. אם הלוקה חיכה מעל 20 שניות ומצב רוחו הנוכחי הוא לא Furious או Angry, השיטה תחזיר מצב רוח Explosive.

ביתן להנich ב mood מטיפוס Mood ו time מטיפוס float

`def __repr__(self)`

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

## המחלקה TypeB

המחלקה TypeB מייצגת אישיות. למחלקה TypeB אין שדות.

ממשו את מחלקה TypeB:

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

שיטה זו מחזירה את מצב הרוח החדש של הלוקו בהתאם במצב הנוכחי שלו וזמן ההמתנה בדרכו. אם הזמן שהלוקו חיכה בדרכו גדול מ-120 שניות ומצב רוחו הנוכחי הוא Furious, השיטה תחזיר מצב רוח Explosive. אם הזמן שהלוקו חיכה בדרכו גדול מ-90 שניות ומצב רוחו הנוכחי הוא Angry, השיטה תחזיר מצב רוח Furious. אם הזמן שהלוקו חERICA בדרכו גדול מ-60 שניות ומצב רוחו הנוכחי הוא Calm, השיטה תחזיר מצב רוח Angry. אם הזמן שהלוקו חERICA בדרכו גדול מ-40 שניות ומצב רוחו הנוכחי הוא Chill, השיטה תחזיר מצב רוח Calm. **ביקט להנich כי mood מטיפוס Mood ו waiting\_time מטיפוס float.**

```
def __repr__(self)
```

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

## המחלקה Angry

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

strength – משתנה מטיפוס int, מספר טبعי גדול מ-0 המייצג את עצמת השפעת מצב הרוח על סבלנות הלוקו.

ממשו את מחלקה Angry:

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

בנאי מקבל את הארגומנט strength ומתחילה את שדות המחלקה. בהדר הארגומנט המתאים יש לאותל את השדה strength עם ערך ברירת המחדל 2. **ביקט להנich כי strength מטיפוס int.**

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

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

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

**ביקט להנich כי waiting\_time מטיפוס float.**

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

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

## המחלקה Furious

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

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

ממשו את מחלקה Furious:

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

בנאי מקבל את הארגומנט strength ומתחילה את שדות המחלקה. בהעדר הארגומנט המתאים יש לאותל 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 כפולת פי 2 מהותוצאה המתתקבלת 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 נפץ ומיוצגת על ידי השדה הבא:

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

ממשו את מחלקה Explosive:

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

בנאי מקבל את הארגומנט strength ומתחילה את שדות המחלקה. בהעדר הארגומנט המתאים יש לאותל 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:

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

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 ומוצגת על ידי השדה הבא:

– משתנה מטיפוס int, מספר טבעי גדול מ-0 המיצג את עוצמת השפעת מצב הרוח על סבלנות הלקות.

ממשו את מחלקה Calm:

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

בנאי מקבל את הארגומנט strength ומתחילה את שדות המחלקה. בהעדר הארגומנט המתאים יש לאותל את השדה strength עם ערך ברירת המחדל 2. קבע להנich בـ strength מטיפוס int.

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

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

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

קבע להנich בـ waiting\_time מטיפוס float.

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

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

## המחלקה Chill

המחלקה Chill מייצגת מצב רוח צוון ומוצגת על ידי השדות הבאים:

– משתנה מטיפוס int, המיצג את עוצמת השפעת מצב הרוח על סבלנות הלקות.

– משתנה מטיפוס float, בין 1 ל-0, המיצג את מידת הרוגע של מצב הרוח.

ממשו את מחלקה Chill:

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

בנאי מקבל את הארגומנטים strength, chill\_modifier ומאתחלת את שדות המחלקה. בהעדר הארגומנטים המתאימים יש לאותל את השדה chill\_modifier עם ערך ברירת המחדל 0.5, וא- 0.5 בהתאם. קבע להנich בـ strength מטיפוס int ו chill\_modifier מטיפוס float.

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

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

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

קבע להנich בـ waiting\_time מטיפוס float.

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

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

## המחלקה ArrivalTimeServingStrategy

המחלקה מייצגת אסטרטגיה של تعدוף הזמן לфи 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 הזמן (מطيיפוס `int`, מספר טבעי גדול מ-0), והערך הוא זוג של לקוחות ומנה (טאפל המכיל מופיע של המחלקה `Customer` ומופיע של המחלקה `Dish` בסדר המצוין). מוחזירה את **מזהה** הזמן הבא על פי זמן ההגעה שלו, במידה ובכמה הזמנות הגיעו באותו הזמן תיבחר הזמן שבסנסה 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 تعدוף הזמן לфи זמן המתנה שלהן.

ממשו את מחלקת LongestWaitingTimeServingStrategy

`def select_next_order(self, orders):`

השיטה מקבלת את המשתנה `orders`, מילון המחזיק את ההזמנות שבוצעו. המפתח הוא מזהה הזמן (מطيיפוס `int`, מספר טבעי גדול מ-0), והערך הוא זוג של לקוחות ומנה (טאפל המכיל מופיע של המחלקה `Customer` ומופיע של המחלקה `Dish` בסדר המצוין). מוחזירה את **מזהה** הזמן שזמן המתנה שלו הוא האחרון ביותר. במידה ובכמה הזמנות מחכחות זמן זהה תיבחר הזמן שבסנסה 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 تعدוף הזמן לфи סבלנותו של הלוקו:

מימוש את מחלקה LeastPatienceCustomerServingStrategy

```
def select_next_order(self, orders):
```

השיטה מקבלת את המשתנה **orders**, מילון המחזיק את הזמנות שבוצעו. המפתח הוא מזהה הזמן (מספר טבעי מ-0), והערך הוא זוג של לקוחות ומינה (טאפל המכיל מופיע של המחלקה Customer ומופיע של המחלקה Dish בסדר המצוין). השיטה מחזירה את מזהה הזמן שהסבלנות של הלוקו שלה היא הנמוכה ביותר. במידה וכמה הזמנות עלות אותו הדבר תבחר הזמן שנכנסה קודם.

במידה ולא קיימות זמנות ב **orders** תזרק שגיאה **OrderOutOfBoundsException**.  
נתן להנich כי orders הוא קלט תקין כפי שתואר.

## המחלקה FalafelStall

מחלקה זו מייצגת דוכן פלאפל שמנהל הזמנות של לקוחות, משרת מנחות, ומנהל את המלאוי והכפסים שלו. המחלקה כוללת רשימה של רכיבים זמינים, מונה הזמנות סטטי, ושיטות שונות לביצוע פעולות כמו הוספה/zمنה, קבלת פרטי הזמן, וסילוק הזמנות.

המחלקה מתוארת ע"י השדות הבאים:  
**orders** - מילון שמחזיק את הזמנות שבוצעו. כל מפתח ב-**orders** הוא מזהה הזמן (מספר טבעי מ-0), והוא ממופה לטאפל המכיל לקוח (לקוח של המחלקה Customer) ומינה (מופיע של המחלקה Dish בסדר המצוין).  
**strategy** - אסטרטגיה לבחירת הזמן הבאה שתוגש. (למשל, לפי סדר ראשון-נכנס-ראשון-ויצא).  
**ingredient\_prices** – מילון מחירים של רכיבים הקיימים בדוכן הפלאפל, כאשר כל מפתח הוא רכיב (מחירו) המופיע למחירו (int או float).  
**money** - משתנה מטיפוס float שמייצג את הסכום המצטבר שהדוכן הרוויח.  
**order\_counter** – מוחזק את כמות הזמנות שבוצעו עד כה בדוכן.

מימוש את המחלקה FalafelStall

```
def __init__(self, strategy, ingredient_prices):
```

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

נתן להנich כי הקלט של השדות תקין.

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

השיטה מקבלת כקלט את `customer` (מופע של המחלקה `Customer`) ומנה (מופע של המחלקה `Dish`).  
מוסיפה הזמנה חדשה למערכת. בלהזמנה חדשה מקבלת מזהה אוטומטי השווה למספר הזמנות שהוזנו עד כה + 1. `order_counter + 1`. במידה וקיים רכיבים במנה שאינם קיימים בדוכן הפלපף תירק שגיאת `NoSuchIngredientException`. על השיטה להחזיר את מזהה הזמנה שנוצרה.

ניתן להניח כי הפעמי `dish` מטיפוס `customer` וכי `customer` מטיפוס `dish`

```
def get_next_order_id(self):
```

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

```
def serve_dish(self, order_id, dish):
```

השיטה מגישה מנתה לפי מזהה הזמנה. השיטה מקבלת כקלט את מזהה את הזמנה (`order_id` משתנה מטיפוס `int`) ואת מנתה הצפוייה (מופע של המחלקה `Dish`) ובודקת אם מנתה שהוגשה תואמת את המנה שהוזמנה. בהתאם לתוצאות הבדיקה השיטה מעדכנת את הרוחחים של הדוכן (בלומר מעדכנת את השدة `money`). אם הזמנה 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 הזמנות לפי מזהה הזמנה. 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`) ומוחזירה את הזמנה לפי המזהה. זורקת חריגת `NoSuchOrderException` אם הזמנה לא קיימת במערכת.  
לא ניתן להניח תקינות הקלט.

```
def calculate_cost(self, dish):
```

השיטה מקבלת כקלט את מנתה (משתנה מטיפוס `Dish`) ועליה להחזיר את המחיר הכללי של מנתה. אם אחד הרכיבים במנה לא נמצא ברשימה הרכיבים הזמינים של הדוכן, נזרקת חריגת `NoSuchIngredientException`.  
ניתן להניח כי `dish` מטיפוס `Dish`

```
def get_orders(self):
```

השיטה מוחזירה את מילון הזמנות.

```
def get_earning(self):
```

השיטה מוחזירה את הסכום המציבר שהדוכן הרוויח.

## המחלקה RandomOrdersStrategy

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

- שדה שתפקידו לעקוב אחר **במota ההזמנות שיוצחו**. משתנה מטיפוס int.
- שדה שתפקידו להציג מקסימום של הזמנות שיוציאו ע"י האיטרטור. ערך שלילי מצין שהאיטרטור ימשך ללא הגבלה (לא סוף). משתנה מטיפוס Int.
- שדה המציג מספר המנות המקסימלי להפקת בכל צעד (אירכזיה), משתנה מטיפוס Int מסpter טבעי הגדל מ-0.
- רשימה של מרכיבים, מהם יבחרו מרכיבים למנה.
- שדה המתאר את מספר הרכיבים המקסימלי שיוכולים להיות במנה. משתנה מטיפוס Int מסpter טבעי הגדל מ-0.

משמעותו של מחלקת RandomOrdersStrategy:

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

בנאי המאתחל את שדות המחלקה בהתאם לארגמנטים שהבנאי מקבל. בהעדר ארגומנט `n_orders` יש לאותלו ב-1.

ניתן להניח כי הקלט של השדות תקין.

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

השיטהמחזירה איטרטור על המחלקה.

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

השיטהמחזירה רשימה של הזמנות `orders` כך שכל הזמנה היא טאף של לקוחות (Customer) ומנות (Dish) בהתאם. במota המנות שתיציר השיטה הוא מספר שלם בין 0 ל-**max\_dishes** ובמota הרכיבים בכל מנה תהיה מספר רנדומאלי בין 1 ל-**max\_ingredients**.

מצב הרוח והאישיות של הלקוח בבחירה באופן רנדומלי בין כל מצב הרוח והאישיות **עם הערכים הדיפולים שלהם**.

שםו של הלקוח הוא שם ייחודי, ונקבע **לפי** במota הזמנות שיוציאו עד כה ועוד 1 (השתמשו ב-`current+1`). **כדי לחת שם ללקוח**.

**current** עוקב אחר **במota ההזמנה שיוצאה בצעד הנוכחי**.

במחלקה זאת תצטרכו להשתמש בספרייה random על מנת לקבל ערכים רנדומאליים. שימושו לב random.choice לפונקציות random.random, random.randomchoice ו-random.chocies.

## המחלקה FixedOrdersStrategy

המחלקה **FixedOrdersStrategy** מייצגת אסטרטגיית ייצור מנוגת קבועה, מתוארת על ידי השדות הבאים:

- שדה מטיפוס רשיימה מקוננת של הזמן, כל איבר ברשימה מכיל רשיימה של הזמן.
- שדה שתפקידו לעקוב אחר התקדמות הנוכחות של האיטרטור. משתנה מטיפוס **int** int ממספר טبعי הגדל מ-0.

מימוש את מחלקה 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):
```

השיטהמחזירה רשיימה של הזמן מ **lst\_orders** לפי **index** הנוכחי שלה.

## המחלקה Game

מחלקה **Game** היא המוח של המשחק ותפקידה לנוהל יצרה של דוכן פלאפל, יצרה של הזמן עבור הדוכן, עדכון מצב הרוח והסבלנות של הלוקותות ובמובן, הכנתן של הזמן ע"י השחקן, **כפי שתואר בהקדמה.**

המחלקה מתוארת על ידי השדות הבאים:

- משתנה המייצג את האסטרטגיה להפקת הזמן (למשל, אקראיות או קבועה).
- משתנה המייצג את האסטרטגיה להגשת הזמן (למשל, לפי זמן הגעה או זמן המתנה).
- מילון מחירים שלרכיבים קיימים בדוכן הפלפל, באשר כל מפתח הוא רכיב (מחוזת) הממופה למחירו (**int** או **float**).
- זמן התחלת המשחק (בעזרת **(int(time.time()))**).
- **game\_start** - מספר הניסיונות של השחקן (התחלת ב-3).
- **lives** - מילון שמכיל את הרכיבים הקיימים 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):`

השיטהמחזירה את מספר הניסיונות של השחקן.

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

השיטה מחשבת ומוחזירה את זמן המשחק (בשניות), לומר כמה זמן עבר מאז שהתחילה המשחק (זמן הנוכחי פחות זמן התחלה). אם הזמן הנוכחי לא מסופק יש לאותל אותו ב `(int(time.time()))`.

ניתן להניח כי current\_time מופיע מטעיפוס float

`def run(self):`

השיטה תאותל אובייקט FalafelStall ותכין הרזנות כל עוד משתמש יש עוד ניסיונות (`lives`) בעזרת אסטרטגיית יצירת הרזנות (`orders_strategy`) השיטה תבצע הרזנות לדוכן הפלפל (יכול להיות כי יוחזרו רשימות ריקות מאסטרטגיית יצירת הרזנות, זהה בסדר, יש להמשיך בתכנית ולהחות עד שיגיעו הרזנות לדוכן).

בשלב זהה, הדוכן מוציא את הרזנות אחת אחרי השניה לפי אסטרטגיית ההגשות (השדה `serving_strategy`) ומציג לשחקן את הרזנה 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 יכנס את הרכיבים שיש למנת מילון הרכיבים, ייצור את המנה (Dish) ויגיש אותה ללקוח.

כדי שהשחקן ייצור את המנה עליו להכניס רצף של מספרים, המופרדים ברווח, המיצגים את המרכיבים של המנה (בסדר נלשהו). לדוגמה, עבור המנה למלטה על השחקן לספק בקלט את המספרים:

1 6 4 6

במידה והוכנס מספר שלא קיים בתפריט (לדוגמה 9) המרכיב ייחס במחוזת ריקה (""). לאחר הזנת המנה הדוכן מגיש את הזמנה ללקוח, אם ההגשה הייתה מוצלת הזמנה נוספת מילון הזמנות של דוכן הפלאף.

שגיאות עולות להתרחש במהלך יצירת הזמנה:

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

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

במידה ונוצרה מנת שלא תואמת את הזמנה של הלוקו יש להדפיס ולהמשיך בתכנית:

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

במידה ולא יהיו הזמנות המחכotta להכנה, יש לחכות עד שתיתבנש הזמנה ללקוח.

בשלב האחרון על השיטה לעדכן את מצב הרוח של הלוקוות שהזמנות שלhn עדין לא הוגשו. במידה והסבלנות של אחד הלוקוות קטנה או שווה ל 0 יש להסיר את הזמנה שלו מהדוכן ולהוריד את כמות הניסיונות של השחקן (lives) ב 1.

המשחק נגמר כאשר הניסיונות של השחקן קטנים או שווים ל 0 ובסיומו תודפס הודעה "Game Over" בצד הציגן הנוכחי (הרוח של הדוכן) והמשחק נעצר:

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

לצורך ריצת התרגיל סיפקנו לכם קוד עק.`main` אשר תוכלו להשתמש בו על מנת להריץ את המשחק לאחר שתיצרו את המחלקות.

## חריגות (Exceptions)

עליכם למשם מספר מחלקות מטיפוס `Exception` בקובץ בשם `uk.exceptions`. במהלך מימוש המשחק יהיה עליכם לזרוק את החיריגות המתאימות בהתאם לדרישות שיפורטו במחלקות האחרות.

### המחלקה NoSuchIngredientException

החריגה תיזורק כאשר מנסים להרכיבמנה מאחד הרכיבים אשר אין נמצאו בדוכן הפלפל. בחריגה זו עליכם לדرس את השיטה `__str__` של המחלקה `NoSuchIngredientException` כך שת>Show More

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

דוגמא:

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

וודפס:

Error:  
"banana" is an invalid ingredient.

### המחלקה NotCustomerDishException

החריגה תיזורק כאשר מנסים להגיש ללקוחמנה שלא תואמת את הזמנה.

בחריגה זו עליכם לדرس את השיטה `__str__` של המחלקה `NotCustomerDishException` כך שת>Show More

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)
```

וודפס:

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

## המחלקה NoSuchOrderException

החריגה תיזרק באשר מנסים לגשת להזמנה שלא קיימת. בחריגה זו עליכם לדחוס 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)
```

וידפס:

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

## המחלקה OrderOutOfBoundsException

החריגה תיזרק באשר מנסים לגשת לרישימת 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 התרגול שלכם בצורה קלה ונוחה תשתמשו בכל בדיקות תבנה שנקרו **unit test**. כל זה הוצג בתרגול מס' 7 ובمعدה מס' 7, וישמש אתכם בהמשך דרככם המקצועית.

עליכם ממש UnitTest לשולשת המחלקות `Calm`, `TypeA` ו- `Dish`, באשר לכל אחת מהמחלקות יש ליצור לפחות 3 טסטים שונים.

כדי לראות כיצד מכינים Unit Test ניתן גם להסתכל 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 מאפשר להריץ `test` על המשחק ע"י הכנסת קלט אוטומטי.