Programowanie I R

Zadania – seria 6.

Programowanie obiektowe – podstawy.

Zadanie 1. `circles` – Porównywanie położeń okręgów.

Niech będzie dany pewien kartezjański układ współrzędnych na płaszczyźnie. Napisz klasę `Circle` reprezentującą okrąg. Klasa powinna zawierać

- pole `x`, przechowujące odciętą środka okręgu w zadanym układzie współrzędnych,
- pole `y`, przechowujące rzędną środka okręgu w zadanym układzie współrzędnych,
- pole `r`, przechowujące promień okręgu,
- metodę `Circumference`, zwracającą obwód okręgu,
- metodę `Intersection`, zwracającą liczbę punktów wspólnych okręgu z innym okręgiem, reprezentowanym przez inną instancję klasy `Circle` przekazaną metodzie `Intersection` jako argument.

Korzystając z tej klasy napisz program `circles`, którego zadaniem jest porównywanie położeń dwóch okręgów. Program powinien przyjmować jako argumenty wywołania sześć liczb zmiennoprzecinkowych opisujących dwa okręgi, oznaczających kolejno: odciętą środka pierwszego okręgu, rzędną środka pierwszego okręgu, promień pierwszego okręgu, odciętą środka drugiego okręgu, rzędną środka drugiego okręgu i promień drugiego okręgu, i wypisywać na standardowe wyjście obwód każdego z tych okręgów oraz liczbę ich punktów wspólnych.

Wskazówka. Nieskończoność może być w języku Python reprezentowana np. jako `float("inf")` lub za pomocą stałej `inf` z modułu `math`. Sprawdzenia, czy zmienna ma wartość równą nieskończoności, można dokonać, korzystając z funkcji `isinf` z modułu `math`.

Zadanie 2. `sortlist` – Lista posortowana.

Napisz klasę `SortedList`, której funkcjonalność będzie podobna do funkcjonalności predefiniowanej listy języka Python – klasa ma zawierać te same metody z wyjątkiem metody `sort` – oferującą dodatkowo automatyczne sortowanie: po każdej zmianie zawartości listy jej elementy powinny być sortowane rosnąco. Wykorzystaj predefiniowaną listę języka Python jako wewnętrzną strukturę danych.

Korzystając z tej klasy napisz program `sortlist`, który przyjmuje jako argumenty wywołania dowolną ilość liczb zmiennoprzecinkowych i wypisuje na ekranie te liczby uporządkowane rosnąco oraz malejąco.

Zadanie 3. `stack` – Implementacja stosu.

Napisz klasę `Stack`, implementującą strukturę danych nazywaną stosem. Wykorzystaj predefiniowaną listę języka Python jako wewnętrzną strukturę danych.

Korzystając z tej klasy napisz program `stack`, który przyjmuje jako argumenty wywołania dowolną ilość liczb zmiennoprzecinkowych i wypisuje na ekranie te liczby w kolejności odwrotnej do tej, w której zostały podane.

Zadanie 4. `epidemic` – Symulacja rozprzestrzeniania się epidemii.

Napisz program `epidemic` implementujący prosty model rozprzestrzeniania się epidemii.

Program ma opisywać niewielką społeczność, której członkowie rozmieszczeni są na planszy o zadanych wymiarach: szerokości w i wysokości h. Na planszy wprowadzamy kartezjański układ współrzędnych (x, y) w taki sposób, aby jego początek pokrywał się z lewym dolnym rogiem planszy, wówczas $x \in [0, w]$ i $y \in [0, h]$. Członkowie społeczności mogą być zdrowi, chorować lub być bezobjawowymi nosicielami choroby. W kolejnych krokach symulacji każdy z nich zmienia swoje położenie, przesuwając się w losowym kierunku o losowy dystans, którego maksymalna wartość powinna być w przypadku osób chorych mniejsza niż w przypadku osób zdrowych i nosicieli. Gdy któraś z osób w wyniku zmiany położenia znajdzie się poza planszą, powinna pojawić się w odpowiednim położeniu z drugiej strony planszy. Jeśli osoba zdrowa znajdzie się odpowiednio blisko osoby chorej lub nosiciela, może się od niej zarazić z pewnym prawdopodobieństwem.

Program powinien definiować dwie klasy:

- klasę `Person`, opisującą pojedynczą osobę, zawierającą:
  - pola `x` i `y`, przechowujące liczby zmiennoprzecinkowe reprezentujące współrzędne osoby na planszy,
  - pole `status`, przechowujące informację o stanie zdrowia osoby (zdrowy, chory, nosiciel), np. w formacie łańcucha tekstowego lub odpowiednio interpretowanej liczby całkowitej,
  - pole statyczne `MaxDistance`, przechowujące maksymalny dystans, jaki może w jednym kroku symulacji przebyć osoba zdrowa lub nosiciel; na początek przypisz mu wartość `1.`,
  - pole statyczne `MaxIllDistance`, przechowujące maksymalny dystans, jaki może w jednym kroku symulacji przebyć osoba chora; na początek przypisz mu wartość `0.1`,
  - metodę `Move`, zmieniającą położenie osoby na planszy o losowy dystans; dystans ten nie powinien być większy od `MaxDistance` lub `MaxIllDistance`, w zależności od wartości pola `status`,
  - metodę `Info`, zwracającą łańcuch tekstowy prezentujący informacje o osobie – jej stan zdrowia i położenie na planszy – w przejrzystym formacie,
  - metodę `__str__` pozwalającą wypisywać informacje o osobie zwracane przez metodę `Info` za pomocą funkcji `print`,
- klasę `Population`, reprezentującą całą modelowaną populację, zawierającą:
  - pole `people`, będące listą zawierającą obiekty klasy `Person` reprezentujące członków populacji,
  - pola `h` i `w`, opisujące, odpowiednio, wysokość i szerokość planszy, na której rozmieszczona jest populacja; na początek przyjmij, że plansza jest kwadratem o boku `100.`,
  - pole statyczne `InfectionProbability`, określające prawdopodobieństwo tego, że na samym początku symulacji konkretna osoba będzie zakażona (tzn. będzie osobą chorą lub nosicielem); na początek przyjmij, że prawdopodobieństwo to wynosi 20%,
  - pole statyczne `InfectionDistance`, określające dystans, na jaki osoba zdrowa musi się zbliżyć do osoby chorej lub nosiciela, by się zarazić; na początek przyjmij, że dystans ten wynosi `1.`,
  - konstruktor przyjmujący jako argumenty wysokość i szerokość planszy oraz liczebność populacji; konstruktor ten powinien umieszczać na liście `people` obiekty klasy `Person` w ilości odpowiadającej liczebności populacji, przypisując każdemu z nich losowe położenie na planszy (tzn. losując wartości pól `x` i `y` każdego z obiektów) oraz losowy stan zdrowia (tzn. losując wartość pola `status` każdego z obiektów); każdy pacjent może otrzymać status zakażonego (a więc nosiciela lub chorego) z prawdopodobieństwem określonym przez wartość pola statycznego `InfectionProbability`, a to, czy osoba zakażona jest nosicielem, czy chorym, powinno być losowane z prawdopodobieństwem 50%,

– metodę `Move`, zmieniającą położenia wszystkich członków populacji, wykorzystując do tego metodę `Move` klasy `People` (czyli wywołując metodę `Move` dla każdego z obiektów na liście `people`); metoda ta musi dbać o to, by osoby, które wyjdą poza planszę, wracały na nią z drugiej strony; po dokonaniu zmiany położeń wszystkich osób metoda ta powinna sprawdzać dystans każdej pary osób, i jeśli jest on mniejszy od `InfectionDistance`, a jedna z osób w parze jest nosicielem lub choruje, druga osoba powinna zachorować lub zostać nosicielem z prawdopodobieństwem 50%,

– metodę `Paint`, wizualizującą rozmieszczenie członków populacji na planszy z wykorzystaniem pakietu *Matplotlib*; osoby zdrowe powinny być zaznaczone na rysunku zielonymi markerami, osoby chore – czerwonymi, zaś nosiciele – żółtymi.

Celem programu jest wyświetlenie animacji, której kolejne klatki rysowane będą za pomocą funkcji `Paint` obiektu reprezentującego populację.

Opracowanie: Bartłomiej Zglinicki.