**Temat 5**

**Zadanie 1.** Lata przestępne

**def** create\_list(first\_year, last\_year):  
 **return** [year **for** year **in** range(first\_year,last\_year+1) **if** (year % 400 == 0) **or** ((year % 4 == 0) **and not** (year % 100 == 0))]

*Uwaga! 2 i 3 linijka są jedną linią tekstu, jest za długa, żeby ją tu poprawnie wkleić.*

Definiuję funkcję **create\_list(first\_year, last\_year),** argumentami są liczby całkowite oznaczające lata.

Lata przestępne będziemy generowali w przedziale **[first\_year;last\_year]**

W ciele funkcja jedynie zwraca list comprehension zawierającą lata przestępne z przedziału.

Składnia list comprehension wygląda następująco:

[wyrażenie **for** obiekt **in** lista **if** warunki ]

W naszym przypadku:

* wyrażenie = **year** -> nie wykonujemy żadnych działań na zmiennej year
* obiekt = year -> **year** jest zmienną sterującą pętli for, zwiększa się o 1
* list = **range(first\_year; last\_year + 1)** -> lista liczb całkowitych z zakresu **[first\_year; last\_year + 1],** w tym zakresie będzie poruszała się zmienna **year**
* warunki = **(year % 400 == 0) or ((year % 4 == 0) and not (year % 100 == 0))** -> liczba dzieli się przez 400 lub dzieli się przez **4** jednocześnie nie dzieląc się przez **100**.

Działa to tak:

Do pustej listy wstawiane są wartości zmiennej **year** przyjmującej wartości całkowite od **first\_year** do **last\_year** jeżeli spełnione są warunki. Rezultatem jest wypełniona lista. Ją właśnie zwraca funkcja.

*Więcej o list comprehension:* [*https://www.pythonforbeginners.com/basics/list-comprehensions-in-python*](https://www.pythonforbeginners.com/basics/list-comprehensions-in-python)

**Zadanie 2.** Pochodna

Implementujemy poniższy wzór w pythonie.

**def** derivative(f, x, h = 0.0001):  
 **return** (f(x+h)-f(x))/h

Przypisanie **0.0001** do **h** w definicji funkcji sprawia, że w razie wywołania funkcji bez argumentu **h** przyjmie ono **0.0001** jako wartość domyślną

Kod:

**def** f(x):  
 **return** x \*\* 2  
  
**def** program():  
 print(derivative(math.sin, 1))  
 print(derivative(math.sin, 0))  
 print(derivative(f, 1, 0.00001))

**Zadanie 3.** Wspólne Elementy

Metoda 1. Z pętlą

**def** loop\_method(list1, list2):  
 result = []  
 **for** element **in** list1:  
 **if** element **in** list2 **and not**(element **in** result):  
 result.append(element)  
 **return** result

Definiuje funkcję przyjmującą dwie listy jako argumenty.

Tworzę pustą listę **result**. Do niej będę dodawał elementy występujące w obu listach.

Tworzę pętle **for**, **element** przybiera kolejne wartości z **list1**.

Sprawdzam, czy wartość zmiennej element należy także do **listy2** oraz nie należy do **result**. Dzieki temu eliminujemy występowanie duplikatów w **result**. Jeśli warunki są spełnione wtedy do listy **result** dodaje **element**.

Funkcja zwraca listę elementów występujących w obu listach wejściowych.

Metoda 2. Operacje na zbiorach

**def** sets\_method(list1, list2):  
 **return** list(set(list1).intersection(set(list2)))

* set(list) zmienia listę w zbiór
* set1.intersection(set2) zwraca zbiór będący iloczynem zbiorów set1, set2
* list(set) zmienia zbiór w listę

Powyższa funkcja zwraca iloczyn zbiorów utworzonych z wejściowych list rzutując go na typ list

**Zadanie 4.** Bisekcja

Tw. Darboux:

*Jeżeli funkcja jest ciągła w przedziale [a;b] to w tym przedziale przyjmuje wszystkie wartości z zakresu [f(a);f(b)].*

Zatem jeśli lub czyli to

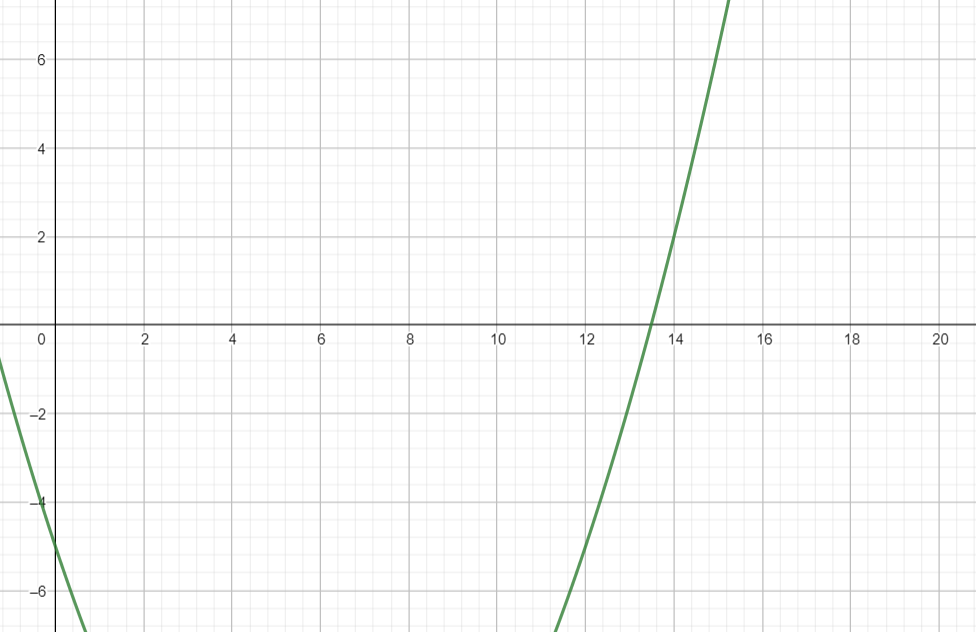
*Jeżeli funkcja jest ciągła i wartość na początku przedziału jest innego znaku niż na końcu przedziału to w tym przedziale jest przynajmniej jedno miejsce zerowe.*

To czy znaki w a i b są różne najwygodniej spradza się warunkiem jeśli jest to prawda to znaki są różne.

Zakładając, że (w przedziale (a;b) występuje miejsce zerowe) oraz to jeśli wartość funkcji na środku tego przedziału (:

1. Jest innego znaku niż a, to miejsce zerowe leży w przedziale
2. Jest innego znaku niż b, to miejsce zerowe leży w przedziale

Rozważmy przykład, szukamy miejsca zerowego poniższego wielomianu w przedziale (0:20)

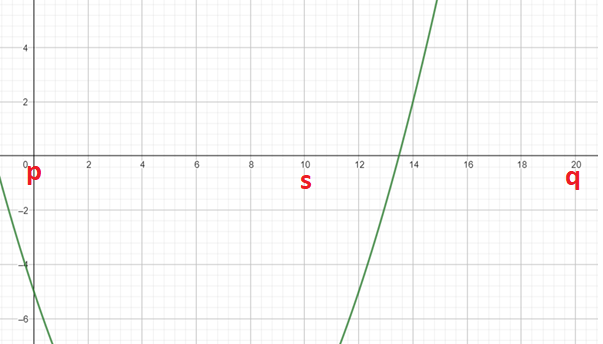


**p** – początek przedziału

**q** – koniec przedzialu

**s** – środek przedziału

Przedział (0;20)

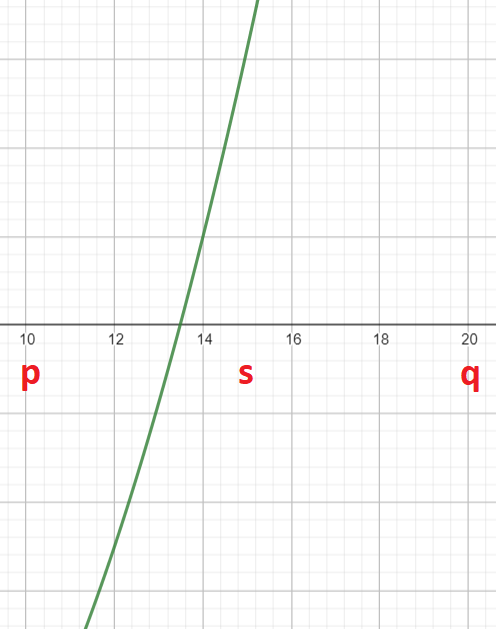
p = 0

q = 20

s = 10

f(s)\*f(p) > 0, więc

p = s = **10**

Przedział (**10**;20)

p = 10

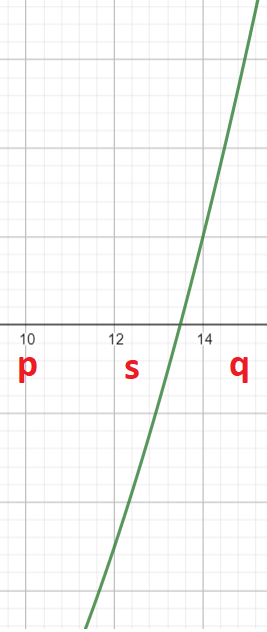
q = 20

s = 15

f(s) \* f(p) < 0, więc

q = s = **15**

Przedział (**15,**20)

p = 10

q= 15

s= 12,5

f(s) \* f(p) > 0, więc

p = s = 12,5

Ważnym spostrzeżeniem jest to, że odległość miejsca zerowego od środka przedziału nigdy nie będzie większa od zatem jeśli poszukujemy miejsca zerowego z dokładnością musimy zawężać przedział dopóki wówczas zwracamy .

Przejdźmy do kodu:

Metoda 1. Rekurencyjna

**def** bisection\_recursive(function, p, q, eps):  
 s = (p + q)/2  
 **if** abs(p - q)/2 < eps **or** function(s) == 0:  
 **return** s  
  
 **if** function(p) \* function(s) < 0:  
 **return** bisection\_recursive(function, p, s, eps)  
 **return** bisection\_recursive(function, s, q, eps)

*Uwaga! Algorytm zaklada, że miejsce zerowe nie leży w p ani q! Lepiej sprawdzić ten warunek przed wywołaniem algorytmu.*

Definiuje funkcje **bisection\_recursive(function, p, q, eps).**

* **function** to funkcja, której miejsc zerowych szukamy.
* **p**- początek przedziału,
* **q**- koniec przedziału
* **eps**- wymagana dokladność

Wyliczam **s** – środek przedziału

Jeżeli przybliżenie jest wystarczające lub **s** jest miejscem zerowym funkcja zwraca **s**, tym samym kończąc działanie.

Jeżeli **p** i **s** są przeciwnych znaków, funkcja wywołuje sie rekurencyjnie dla przedziału **(p,s)**

W innym przypadku (**p** i **s** tego samego znaku, **s** i **q** różnych znaków) funkcja wywołuje sie rekurencyjnie na przedziale **(s,q)**

Ponieważ to pewnym jest, że w końcu zajdzie warunek graniczny i rekurencja zakończy się zwracając prawidłowy wynik.

Metoda 2. Pętla while

**def** bisection\_while(function, p, q, eps):  
 s = (p + q) / 2  
 **while** abs(p - q)/2 > eps **and** function(s) != 0:  
 s = (p + q) / 2  
 **if** function(p) \* function(s) < 0:  
 q = s  
 **else**:  
 p = s  
 **return** (p + q) / 2

Definiuje funkcje bisection\_while(function, p, q, eps).

* function to funkcja, której miejsc zerowych szukamy.
* p- początek przedziału
* q- koniec przedziału
* eps- wymagana dokladność

Wyliczam **s** – środek przedziału

Tworzę pętle **while** działającą dopóki dokładność przybliżenia jest zbyt mała i miejsce zerowe nie leży dokładnie na środku przedziału.

W każdym obiegu pętli wyliczam środek przedziału.

Sprawdzam czy **p** i **s** są przeciwnych znaków, w zależności od tego ustwiam **q** = **s** lub **p** = **s**

W momencie kiedy **p** i **q** są wystarczająco “blisko siebie”, aby maksymalny błąd był mniejszy niż **eps** petla kończy działanie.

Funkcja zwraca środek przedziału, dla **p** i **q** wyliczonych w ostatnim obiegu pętli

**Zadanie 5.** Dekorator

**import** time  
  
**def** dekorator(function):  
 start\_time = time.clock()  
 function()  
 end\_time = time.clock()  
 print(end\_time-start\_time)

Importuje moduł **time** do wyliczania czasu.   
Definiuje funkcję **dekorator**.  
Zapisuje aktualny czas w zmiennej **start\_time**.  
Uruchamiam “udekorowaną” funkcję.  
Po zakończeniu działania funkcji zapisuje aktualny czas w zmiennej **end\_time**.

Wypisuje różnicę czasów **strat\_time** i **end\_time**.

*Więcej o dekoratorach:* [*https://chyla.org/blog/Python\_-\_Dekoratory/*](https://chyla.org/blog/Python_-_Dekoratory/)

*Wykorzystując napisany dekorator można porównać czasy w których zadziała implementacja Zadania 5 z Tematu 3 w wersji z Sitem Euklidesa i testami Millera-Rabina. Polecam przetestować dla n=100000, n=1000000*