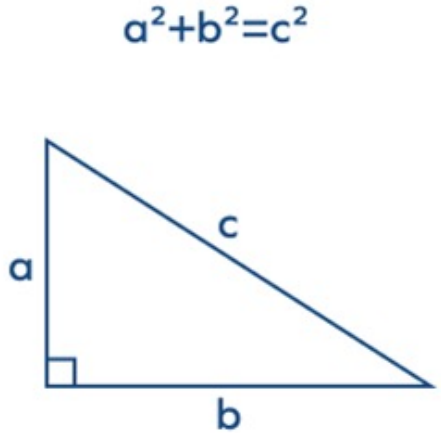
ΑΣΚΗΣΕΙΣ ΠΡΟΟΔΟΥ ΕΡΓΑΣΤΗΡΙΟΥ

2023-2024

Ά Σ Κ Η Σ Η # 1



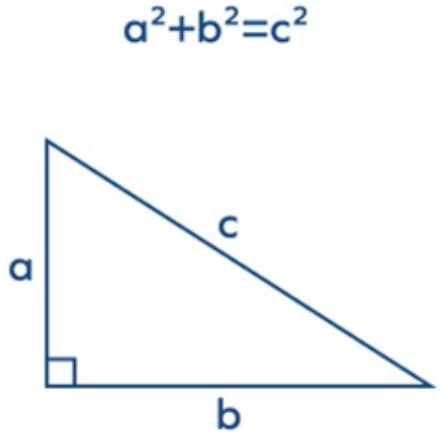
"Πυθαγόρεια Τριάδα" ονομάζεται μια τριάδα φυσικών αριθμών (a, b, c), όπου 0 < a < b < c και $\mathbf{a^2 + b^2 = c^2}$.
Για παράδειγμα, η τριάδα (3, 4, 5) είναι πυθαγόρεια, καθώς $3^2 + 4^2 = 5^2$. Το άθροισμα των στοιχείων της (3, 4, 5) είναι 3 + 4 + 5 = 12. Το γινόμενο των στοιχείων της (3, 4, 5) είναι 3 * 4 * 5 = 60.

Γράψτε συνάρτηση Python που δέχεται ως όρισμα έναν φυσικό αριθμό και επιστρέφει μια **λίστα με τις πυθαγόρειες τριάδες** που έχουν άθροισμα τον αριθμό αυτό.

Στη συνέχεια γράψτε πρόγραμμα Python που καλεί τη συνάρτηση αυτή και τυπώνει για κάθε πυθαγόρεια τριάδα, την τριάδα και το γινόμενο των στοιχείων της. Ελέγξτε τη συμπεριφορά του προγράμματός σας για την/τις τριάδα/ες με άθροισμα 12.

Στη συνέχεια με χρήση του προγράμματος αυτού βρείτε την πυθαγόρεια τριάδα με άθροισμα στοιχείων **1000** (υπάρχει μόνο μία) και τυπώστε το γινόμενο των ψηφίων της. Γράψτε αυτό το γινόμενο ως απάντηση σε αυτό το ερώτημα.

Μετρηση χρονου εκτελεσης



Η συνάρτηση time.process_time() επιστρέφει σε δευτερόλεπτα το χρόνο χρήσης της CPU του προγράμματός μας – καλείται στην αρχή και στο τέλος του προγράμματος

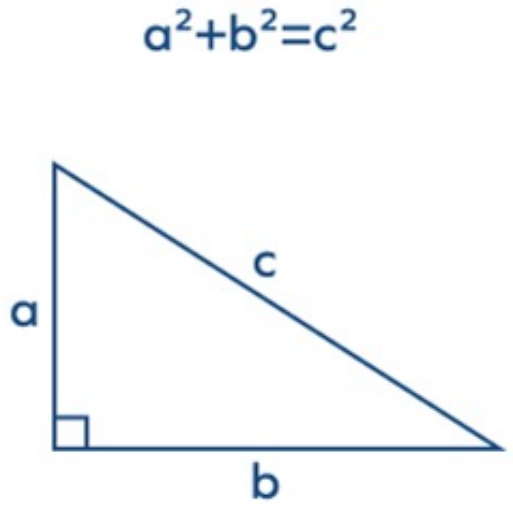
```
import time

# συνάρτηση εύρεσης πυθαγόρειων τριάδων
def pythagorian(n):
    t_start = time.process_time()

    # λύσε το πρόβλημα

    t_process_ms = 1000 * (time.process_time() - t_start)
    return απάντηση, f"{t_process_ms:.3f}"
```

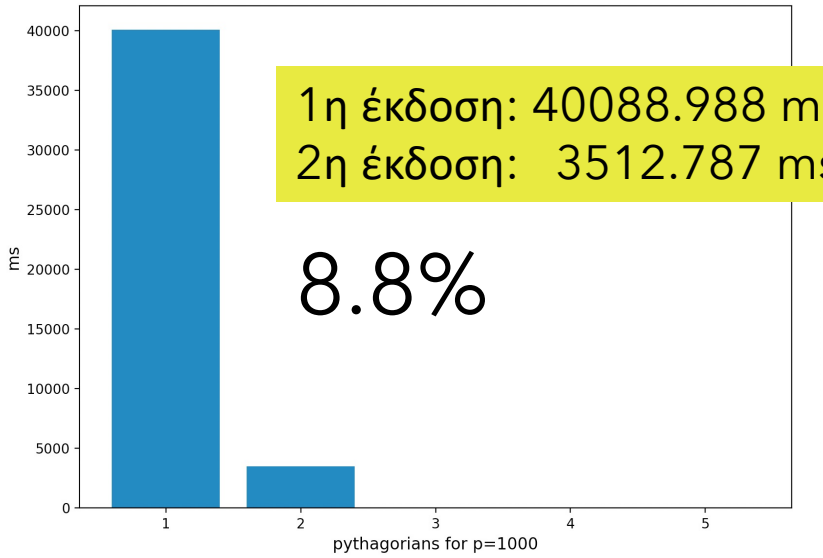
ΛΥΣΗ 1Η




```
def pythagorian1(n):
    t_start = time.process_time()
    triads = []
    for a in range(1,n):
        for b in range(1, n):
            for c in range(1, n):
                if a+b+c == n and a**2 + b**2 == c**2:
                    triads.append((a,b,c))
    t_process_ms = 1000 * (time.process_time() - t_start)
    return triads, f"{t_process_ms:.3f}"
```

Η λύση αυτή έχει αρκετά προβλήματα

ΛΥΣΗ 2Η



1η έκδοση: 40088.988 ms
2η έκδοση: 3512.787 ms

8.8%

```
def pythagorian2(n):
    t_start = time.process_time()
    triads = []
    for a in range(1,n-1):
        for b in range(a+1, n-1):
            for c in range(b+1, min(n-1, a+b):
                if a+b+c == n and a**2 + b**2 == c**2:
                    triads.append((a,b,c))
    t_process_ms = 1000 * (time.process_time() - t_start)
    return triads, f"{t_process_ms:.3f}"
```

βελτίωση της λύσης με αυτές τις αλλαγές – λιγότερο από 1 δέκατο του χρόνου

ΛΥΣΗ 3Η

2η έκδοση: 3512.787 ms
3η έκδοση: 44.484 ms

1,3%

```
def pythagorian3(number):
    t_start = time.process_time()
    triples_list = []
    for a in range(1,number):
        for b in range(1,a):
#Defining c based on number: number == a + b + c
            c = number - (a+b)
            if c*c == a*a + b*b and c > b:
                triples_list.append((a,b,c))
    total_time = 1000 * (time.process_time() - t_start)
    return triples_list, f"{total_time:.3f} ms"
```

2 μόνο βρόχοι, αφού για **a,b** υπολογίζουμε το **c**

ΛΥΣΗ 4Η

3η έκδοση: 44.484 ms
4η έκδοση: 0.446 ms

1,0%

```
def pythagorian4(p):
    t_start = time.process_time()
    triples_list = []
    for b in range(1,p):
        a = int(p - b + ((-p * p + 2 * p * b - 2 * b * b)/(2 * p - 2 * b)))
        c = int((-p * p + 2 * p * b - 2 * b * b)/(-2 * p + 2 * b))
        if a * a + b * b == c * c and a + b + c == p and c > b > a > 0 :
            triples_list.append((a,b,c))
    total_time = 1000 * (time.process_time() - t_start)
    return triples_list, f"{total_time:.3f} ms"
```

1 μόνο βρόχος, αφού για κάθε **b** υπολογίζουμε **a,c**

ΛΥΣΗ 5Η

4η έκδοση: 0.446 ms
5η έκδοση: 0.222 ms

```
def pythagorian5(p):
    t_start = time.process_time()
    triples_list = []
    for b in range(1,p):
        c = int(( 0.5 * p * p + b * b - p * b)/(p - b))
        a = p - b - c
        if a * a + b * b == c * c and c > b > a > 0 :
    triples_list.append((a,b,c))
    total_time = 1000 * (time.process_time() - t_start)
    return triples_list, f"{total_time:.3f} ms"
```

50,0%

απλοποίηση των πράξεων της λύσης 4,

ΛΥΣΗ 6Η

5η έκδοση: 0.222 ms
6η έκδοση: 0.009 ms

4,0%

```
def pythagorian6(p):
    t_start = time.process_time()
    tr=[]
    #a=r^2-s^2, b=2rs, c=r^2+s^2,a+b+c=2*r*s+2*r^2
    for r in range(1,1+int((p//2)**0.5)):
    #a+b+c<n -> 2*r^2<n -> r<(n/2)^(1/2)
        s=(p-2*(r**2))/(2*r)
        #a+b+c=2*r*s+2*r^2 ->s=(n-2*(r^2))/(2*r)
        if(s.is_integer() and s<r):
            tr.append((r**2-s**2,2*r*s,r**2+s**2))
    t_process_ms=1000*(time.process_time()-t_start)
    return tr, f"{total_time:.3f} ms"
```

λύση με βάση τον τύπο του Ευκλείδη

Σ Υ Μ Π Ε Ρ Α Σ Μ Α

- Παρατηρούμε ότι με διαδοχικές διαφορετικές προσεγγίσεις μειώνεται ο χρόνος εκτέλεσης μιας διαδικασίας υπολογισμού των πυθαγόρειων τριάδων για ένα τρίγωνο περιμέτρου p = 1000
- Συνολικά μεταξύ της 1ης και 6ης λύσης η βελτίωση είναι της τάξης 0.0002%, από 40 sec σε 0.009 χιλιοστά του sec.
- Η διαδικασία επαναλαμβάνεται 10 φορές ώστε να πάρουμε τη μέση τιμή του χρόνου εκτέλεσης

```
pythagorians = {1: pythagorian1, 2:
pythagorian2, 3: pythagorian3,
4: pythagorian4, 5: pythagorian5,
6: pythagorian6}

for i,f in pythagorians.items():
    t = tester(f, 1000)
    print(f"{i}η έκδοση: {t:.3f}")
```

```
def tester(f, value=1000):
    result = []
    for _i in range(10):
        r, t = f(value)
        result.append(float(t.strip(" ms")))
    return sum(result)/len(result)
```