Miller-Rabin Primality Test

Miller-Rabin primality test is an optimization of Fermat primality test (based on Fermat's little theorem).

Full Implementation Codes:

```
import random

"""
This function is called for all k trials. It returns False if n is composite
and returns True if n is probably prime.
d is an odd number such that d * 2 ^ r = n - 1 for some r >= 1
"""
def millerTest(d, n):
    # Pick a random number in [2, n-2]
    # Corner cases in isPrime function make sure that n > 4
    a = 2 + random.randint(1, n - 4)

    # Compute a ^ d % n
    x = pow(a, d, n)

    if (x == 1 or x == n - 1):
        return True

    """
    Keep squaring x while one of the following does not happen
    (1) d does not reach n - 1
    (2) (x ^ 2) % n is not 1
    (3) (x ^ 2) % n is not n - 1
    """
    while (d != n - 1):
        x = (x * x) % n
        d *= 2

        if (x == 1):
            return False
        if (x == n - 1):
            return True

    # If no x satisfies, n is a composite
    return False

"""
It returns False if n is composite and returns True if n is probably prime
(pseudoprime).
k is an input parameter that determines accuracy level. Higher level of k
indicates more accuracy.
"""
def isPrime(n, k):

    # Corner cases
    if (n <= 1 or n == 4):
        return False
```

```
    if (n <= 3):
        return True

    # Find r such that n = 2 ^ s * d + 1 for some d >= 1
    # d is an odd number
    d = n - 1
    while (d % 2 == 0):
        d //= 2

    # Iterate given number of "k" times
    for i in range(k):
        if (millerTest(d, n) == False):
            return False
    return True

"""
Main programme
"""
def main():
    # Number of iterations
    k = 4

    upperBound = int(input("Find all primes below: "))
    print(f"All primes smaller than {upperBound}: ")
    print()
    counter = 0
    for n in range(1, upperBound):
        if (isPrime(n, k)):
            print(n, end=" ")
            counter += 1
    print("\n")
    print(f"{counter} primes in total")
    print("\n" * 3)

main()
```

Mathematical Theories:

Fermat's Little Theorem:

$$a^{n-1} \equiv 1 \bmod n$$

(n is a prime, a is an integer that $1 < a < n-1$)

$$\because n \text{ is a prime}$$

$$\therefore n-1 \text{ is an even number}$$

$$\text{let } n-1 = 2^r \times d \ [1]$$

(d is an odd number, which chould not be further divided by 2)

$$\therefore a^{n-1} \equiv 1 \bmod n$$

$$\Rightarrow a^{n-1} - 1 \equiv 0 \bmod n$$

$$\Rightarrow a^{2^s \times d} - 1 \equiv 0 \bmod n$$

$$\Rightarrow (a^{2^{s-1} \times d} - 1)(a^{2^{s-1} \times d} + 1) \equiv 0 \bmod n$$

$$\Rightarrow (a^{2^{s-1} \times d} + 1)(a^{2^{s-2} \times d} + 1) \cdots (a^d + 1)(a^d - 1) \equiv 0 \bmod n$$

$$\Rightarrow (a^{2^{s-1} \times d} + 1 \equiv 0 \bmod n) \vee (a^{2^{s-2} \times d} + 1 \equiv 0 \bmod n) \vee \cdots \vee (a^d + 1 \equiv 0 \bmod n) \vee (a^d - 1 \equiv 0 \bmod n)$$

(if any of these is true, n is a pseudoprime, if none of these is true, n is a composit)

Back to the Python Codes:

1. Firstly, let d = n -1. Keep taking out 2 from d, until d is an odd number (to fulfill the format of [1]).

```
# Find r such that n = 2 ^ s * d + 1 for some d >= 1
# d is an odd number
d = n - 1
while (d % 2 == 0):
      d //= 2
```

2. Specify the number of iterations. The Miller-Rabin primality test can find pseudoprimes, which means there is a possibility that the number found is a composite. Repeating the algorithm with different random value of a will increase the accuracy, but take longer time.

```
# Number of iterations
k = 4
```

3. Generate random number a according to the requirement in Fermat's little theorem

```
# Pick a random number in [2, n-2]
# Corner cases in isPrime function make sure that n > 4
a = 2 + random.randint(1, n - 4)
```

4. Verify if $a^d + 1 \equiv 0 \bmod n$ or $a^d - 1 \equiv 0 \bmod n$.

   $$\text{If } a^d \bmod n = 1, (a^d - 1) \bmod n = 0$$

   $$\text{If } a^d \bmod n = n - 1, (a^d + 1) \bmod n = 0$$

```python
# Compute a ^ d % n
x = pow(a, d, n)
if (x == 1 or x == n - 1):
    return True
```

5. If first 2 cases not satisfied, keep trying the rest, until d = n – 1 (the case $a^{2^s \times d} + 1 \equiv 0 \bmod n$).

   $$x = a^d \bmod n$$

   $$x' = x^2 \bmod n = (a^d \bmod n)^2 \bmod n = a^{2d} \bmod n$$

   (see Diffie-Hellman Key Exchange file for proof)

   $$\text{If } a^{2^k \times d} \bmod n = n - 1, (a^{2^k \times d} + 1) \bmod n = 0$$

```python
while (d != n - 1):
    x = (x * x) % n
    d *= 2

    if (x == 1):
        return False
    if (x == n - 1):
        return True

# If no x satisfies, n is a composite
return False
```

Running Outcome:

```
PS D:\Programming\Python\Cryptography in Python>  & 'C:\Users\sunny\AppData\Local\Microsoft\WindowsApps\python3.10.exe' 'c:\Users\sunny\.vsco
de\extensions\ms-python.python-2022.14.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '59563' '--' 'd:\Programming\Python\C
ryptography in Python\MillerRabin.py'
Find all primes below: 10000
All primes smaller than 10000:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 19
1 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397
 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617
619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 8
59 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 10
87 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 128
9 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489
 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697
1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1
931 1933 1949 1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081 2083 2087 2089 2099 2111 2113 2129 2131 21
37 2141 2143 2153 2161 2179 2203 2207 2213 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293 2297 2309 2311 2333 2339 2341 2347 2351 235
7 2371 2377 2381 2383 2389 2393 2399 2411 2417 2423 2437 2441 2447 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551 2557 2579 2591 2593
 2609 2617 2621 2633 2647 2657 2659 2663 2671 2677 2683 2687 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791
2797 2801 2803 2819 2833 2837 2843 2851 2857 2861 2879 2887 2897 2903 2909 2917 2927 2939 2953 2957 2963 2969 2971 2999 3001 3011 3019 3023 3
037 3041 3049 3061 3067 3079 3083 3089 3109 3119 3121 3137 3163 3167 3169 3181 3187 3191 3203 3209 3217 3221 3229 3251 3253 3257 3259 3271 32
99 3301 3307 3313 3319 3323 3329 3331 3343 3347 3359 3361 3371 3373 3389 3391 3407 3413 3433 3449 3457 3461 3463 3467 3469 3491 3499 3511 351
7 3527 3529 3533 3539 3541 3547 3557 3559 3571 3581 3583 3593 3607 3613 3617 3623 3631 3637 3643 3659 3671 3673 3677 3691 3697 3701 3709 3719
 3727 3733 3739 3761 3767 3769 3779 3793 3797 3803 3821 3823 3833 3847 3851 3853 3863 3877 3881 3889 3907 3911 3917 3919 3923 3929 3931 3943
3947 3967 3989 4001 4003 4007 4013 4019 4021 4027 4049 4051 4057 4073 4079 4091 4093 4099 4111 4127 4129 4133 4139 4153 4157 4159 4177 4201 4
211 4217 4219 4229 4231 4241 4243 4253 4259 4261 4271 4273 4283 4289 4297 4327 4337 4339 4349 4357 4363 4373 4391 4397 4409 4421 4423 4441 44
47 4451 4457 4463 4481 4483 4493 4507 4513 4517 4519 4523 4547 4549 4561 4567 4583 4591 4597 4603 4621 4637 4639 4643 4649 4651 4657 4663 467
8941 8951 8963 8969 8971 8999 9001 9007 9011 9013 9029 9041 9043 9049 9059 9067 9091 9103 9109 9127 9133 9137 9151 9157 9161 9173 9181 9187 9
199 9203 9209 9221 9227 9239 9241 9257 9277 9281 9283 9293 9311 9319 9323 9337 9341 9343 9349 9371 9377 9391 9397 9403 9413 9419 9421 9431 94
33 9437 9439 9461 9463 9467 9473 9479 9491 9497 9511 9521 9533 9539 9547 9551 9587 9601 9613 9619 9623 9629 9631 9643 9649 9661 9677 9679 968
9 9697 9719 9721 9733 9739 9743 9749 9767 9769 9781 9787 9791 9803 9811 9817 9829 9833 9839 9851 9857 9859 9871 9883 9887 9901 9907 9923 9929
 9931 9941 9949 9967 9973

1229 primes in total
```

Although “pseudoprime” sounds not very accurate, the result is actually very reliable. In the case above, k is set to 4.