Exercises for class, 1st week, Course T2

```
print("Hello, " + "World")
print("Hello, ", "World")
```

```
for j in [1, 2, 3]:
    for l in [1, 2, 3]:
        print(j)

```

```
a = 2
for j in range(3):
    a = j*a
print(a)
```

"Infinite" sums/products

Simplest sum

What is the sum

$$1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\ldots=?$$

Calculate the truncated sum, i.e.,

$$\sum_{k=0}^{n} 2^{-k}$$

```
my_sum = 0
for k in range(50):
    my_sum = my_sum + 2**-k
    print(F"k={k:02}: {my_sum:.10g}")
    #print(k, my_sum)
    #print("k = %2d  sum = %.20f"%(k, my_sum))
```

```
# we can also save the list of new terms in the list while summing
powers = []
for k in range(40):
    powers.append((1/4)**k)
    print(F"k={k:02}: {sum(powers):.25g}", powers)

#print(F"evaluated directly:\n      {4/3:.25g}")
```

```
# let us make this code more universal by introducing a new variable
q=2/3
s=0
for k in range(20):
    s+=q**k
    print(F"k={k:02}: {s:.25g}")

print(F"evaluated directly:\n      {1/(1-q):.25g}")
```

Approximations of π

We look at series that can be used to approximate the irrational number π.

$$\pi = \lim_{n\to\infty}\sum_{k=0}^{n}\frac{(-1)^k}{4^k}\left(\frac{2}{4k+1}+\frac{2}{4k+2}+\frac{1}{4k+3}\right)$$

$$\pi = \lim_{n\to\infty}\sum_{k=0}^{n}\frac{1}{16^k}\left(\frac{4}{8k+1}-\frac{2}{8k+4}-\frac{1}{8k+5}-\frac{1}{8k+6}\right)$$

$$2\pi = \lim_{n\to\infty}\sum_{k=0}^{n}\frac{1}{16^k}\left(\frac{8}{8k+2}+\frac{4}{8k+3}+\frac{4}{8k+4}-\frac{1}{8k+7}\right)$$

$$\frac{\pi^2}{6} = \lim_{n\to\infty}\sum_{k=1}^{n}\frac{1}{k^2}$$

$$\frac{\pi^4}{90} = \lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k^4}$$

Your task is to evaluate these sums for small values of n, and observing whether the estimate obtained from them converges to π, and if so, how fast.

The value of π up to 20 decimals is 3.14159265358979311599796346854 4. However, python uses so-called "double precision" numbers which are precise up to 16 digits.

A little help, on outputting the difference between the estimate pi_1 and the supposed limit pi:

```
print(pi_1, "{:+.0e}".format(pi-pi_1))
```

```
# prompt: generate pi**4 using infinite sum

import math

def pi_fourth_sum(n):
  """Calculates an approximation of pi^4/90 using the infinite sum."""
  total = 0
  for k in range(1, n + 1):
    total += 1 / (k ** 4)
  return total * 90

# Calculate the approximation for a few values of n
for n in [10, 100, 1000, 10000]:
  result = pi_fourth_sum(n)
  pi_approx = math.sqrt(result * 6)  # Calculate pi from pi^4/90
  print(f"n = {n}, pi^4/90 = {result:.10f}, pi approximation = {pi_approx:.10f}")

```

```
# 4th formula: print the square root of 6*sum to compare it to pi
my_sum = 0
for k in range(1, 201):
    my_sum += k**-2
    # we need many terms so only every 20th truncated sum is printed
    if k % 20 == 0:
        print(F"k={k:02}: {(6*my_sum)**.5:.25g}")
        print("k={:2d}:  {:.25g}".format(k, (6*my_sum)**0.5))
        print("k=%2d: %.25g"%(k, (6*my_sum)**0.5))

# print(F"evaluated directly:\n      {4/3:.25g}")
```

A szerkesztéshez kattintson duplán (vagy használja az Enter billentyűt)

```
pi = 3.14159265358979311599796346854 4
print("{:+.20f}".format(pi))

max_iter = 20

print(" Sum 1: ")

pi_1 = 0
for k in range(max_iter):
    pi_1 += (-1)**k / 4**k * (2/(4*k+1) + 2/(4*k+2) + 1/(4*k+3))
    print("k={:02}".format(k), pi_1, "{:+.0e}".format(pi-pi_1))

print(" ")
print(" Sum 2, Plouffe: ")

pi_1 = 0
for k in range(max_iter):
    pi_1 += 1 / 16**k * (4/(8*k+1) - 2/(8*k+4) - 1/(8*k+5) - 1/(8*k+6))
    print("k={:02}".format(k), pi_1, "{:+.0e}".format(pi-pi_1))

print(" ")
print(" Sum 3: ")

pi_1 = 0
for k in range(max_iter):
    pi_1 += 0.5 / 16**k * (8/(8*k+2) + 4/(8*k+3) + 4/(8*k+4) - 1/(8*k+7)
    print("k={:02}".format(k), pi_1, "{:+.0e}".format(pi-pi_1))

print(" ")
print(" 1/k^2: ")

sum_1 = 0
```

```
for k in range(1, max_iter+1):
    sum_1 += 1/(k*k)
    pi_1 = (6*sum_1)**0.5
    print("k={:02}".format(k), pi_1, "{:+.0e}".format(pi-pi_1))

print(" ")
print(" 1/k^4: ")

sum_1 = 0
for k in range(1, max_iter+1):
    sum_1 += 1/(k*k*k*k)
    pi_1 = (90*sum_1)**0.25
    print("k={:02}".format(k), pi_1, "{:+.0e}".format(pi-pi_1))

```

EXTRA: Wallis product to calculate π

Modify the above code to calculate π as a product (see Wallis product):

$$\pi = 2\prod_{n=1}^{\infty} \frac{4n^2}{4n^2-1}$$

[ ] ↳ *1 cella elrejtve*

Iterations and conditions

Collatz conjecture.

The Collatz conjecture is one of the most famous unsolved problems in mathematics. The conjecture asks whether repeating two simple arithmetic operations will eventually transform every positive integer into 1. It concerns sequences of integers in which each term is obtained from the previous term as follows: if the previous term is even, the next term is one half of the previous term. If the previous term is odd, the next term is 3 times the previous term plus 1. The conjecture is that these sequences always reach 1, no matter which positive integer is chosen to start the sequence.

Write a python function that prints the sequence up to at most 1000 terms, for any integer used as a starting number.

```
# this is a definition of a function
def collatz(number):
    print(number)
    i=0
    i_max=1000
    while i<i_max and number!=1:
        if number%2==0: number=number//2
        else: number = 3*number+1
        print(number)
        i+=1
    if (i==i_max):
        print(f"Hey, we have still not landed on 'i' after {i_max} steps!")

collatz(121)
```

```
# prompt: cenerate numbers  with collatz conjecture

def collatz(number):
    print(number)
    i = 0
    i_max = 1000
    while i < i_max and number != 1:
        if number % 2 == 0:
            number = number // 2
        else:
            number = 3 * number + 1
        print(number)
        i += 1
    if (i == i_max):
        print(f"Hey, we have still not landed on 'i' after {i_max} steps!")

collatz(121)

```

Primes

How to check if number a is a prime?

```
a=127
divisor_found=False             # divisor is not found yet
for i in range(2,1+round(a**.5)):   # it is enough to check divisors up to the square root of 'a' with some "safety overh
  if a%i==0:
    divisor_found=True
    break   # this will break out of the "for" cycle at once a divisor is found

if divisor_found:
  print(F"{a} is not a prime: {i} divides it.")
else:
  print(F"{a} is a prime.")

```

Let us plug the above code into another larger 'for' cycle over a

```
for a in range(2,12):
  divisor_found=False             # divisor is not found yet
  for i in range(2,1+round(a**.5)):   # it is enough to check divisors up to the square root of 'a' with some "safety ove
    if a%i==0:
      divisor_found=True
      break   # this will break out of the "for" cycle at once a divisor is found

  if divisor_found:
    print(F"{a} is not a prime: {i} divides it.")
  else:
    print(F"{a} is a prime.")
```

Let us search & list twin primes. We could in principle use the same code but let us rather **define a function** to check if a number is a prime.

```
# input should be integer (we do not check it)
# output is bool: True if num is prime
def is_prime(num):
  if num < 2: # by def 0,1 are not prime
    return False
  divisor_found=False             # divisor is not found yet
  for i in range(2,1+round(num**.5)):   # it is enough to check divisors up to the square root of 'a' with some "safety c
    if num%i==0:
      divisor_found=True
      break   # this will break out of the "for" cycle at once a divisor is found

  return not divisor_found

for a in range(25):
  if not is_prime(a):
    print(F"{a} is not a prime.")
  else:
    print(F"{a} is a prime.")
```

Once defined, the same `is_prime()` function can be reused anywhere. The same list can be printed in a more concise way using Python's ternary if-else operator.

```
for a in range(25):
  print(F"{a} is {'' if is_prime(a) else 'not'} a prime.")
```

Now we can easily find & list **twin primes**. We know that we need to check neighboring odd numbers so the simplest (but by far not the most optimal) solution is:

```
for a in range(3,400,2): # we search over only odd numbers
  if is_prime(a) and is_prime(a+2): print(F"Twin primes found: {a}, {a+2}")
```