Oppgave 1: Tallrepresentasjon

For hvilke av disse uttrykkene (skrevet i Python), vil resultatet bli representert som heltall, eksakt flyttall, ikke eksakt i Python, eller gi en feil?

Hvis man skriver uttrykket i venstre kolonne, så velg "Heltall" hvis tallet lagres som heltall, "Eksakt flyttall" hvis det lagres som flyttall med den matematiske eksakte verdien til uttrykket, "Ikke eksakt flyttall" hvis det lagres som flyttall, og "error" hvis uttrykket vil gi en feilmelding.

Oppgave 9: Numerikk.

- 0.0 og 1.0 er verdier som kan representeres eksakt som flyttall.
- `math.pi \approx 3.141592653589793` er ikke eksakt lik $\pi =$.
- Forskjellen mellom disse er omtrent $1.2e-16$
- Siden $\sin(0) = 0$ og $\cos(0) = 1$, så vil Python gi eksakte verdier for disse.
- $\cos(\pi) = -1$ eksakt, mens `math.cos(math.pi)` gir det representerbare tallet nærmest cosinus av `math.pi`, som er -1.0. Vi har mye mindre presisjon rundt -1.0 enn vi har rundt 0.0, og i tillegg er $\cos(x)$ nesten flat rundt $x = \pi$.
- Det samme gjelder for `math.sin(math.pi/2)=1.0`.
- `math.sin(math.pi)` gir 1.2246467991473532e-16, som jo er avstanden mellom π og `math.pi`. Rundt null har vi mye høyere presisjon enn rundt π, så vi kan representere dette nøyaktig. ($\sin(x) har stigningstall -1 i $x = \pi$).

```python
## Oppgave 14: List to Dict

def list_to_dict(input_list, empty_element):
    output_dict = {}
    for i, item in enumerate(input_list):
        if item != empty_element:
            output_dict[i] = item
    return output_dict
```

Python

```python
## Oppgave 15: Sykler

## NB! LF er langt fra den raskeste løsningen, siden det å lete
# i en liste er relativt tregt, og vi må gjøre det i hver iterasjon
def find_cycle(f, start_value, max_length):
    visited = []
    current = start_value
    for i in range(max_length):
        j = visited.index(current) if current in visited else -1
        if j != -1:
            return current, j, i-j
        visited.append(current)
        current = f(current)
    return None

# Her bruker vi en ordbok for å holde styr på hvilke verdier vi har sett før
# og utnytter at den mer effektivt bruker lister for å finne ut om vi har sett
# en verdi før
def find_cycle_faster(f, start_value, max_length):
    visited = {}
    current = start_value
    for i in range(max_length):
        if current in visited:
            return current, visited[current], i - visited[current]
        visited[current] = i
        current = f(current)
    return None
```

Python

```python
## Tester for koden over:

def test_find_cycle():
    f = lambda x: (x * 2) % 9
    assert find_cycle(f, 1, 10) == (1, 0, 6)
    assert find_cycle_faster(f, 1, 10) == (1, 0, 6)
    assert find_cycle(f, 1, 4) == None
    assert find_cycle_faster(f, 1, 4) == None


    f = lambda x: x//2 if x % 2 == 0 else (x * 3 + 1)
    assert find_cycle(f, 1, 10) == (1, 0, 3)
    assert find_cycle_faster(f, 1, 10) == (1, 0, 3)

    print("Alle tester for find_cycle() bestått!")

test_find_cycle()
```

Python

```
Alle tester for find_cycle() bestått!
```

```python
import math
print(math.pi)

from decimal import Decimal

print(Decimal(math.pi))

print(math.sin(math.pi))
```

Python

```
3.141592653589793
3.141592653589793115997963468544185161590576171875
1.2246467991473532e-16
```

```python
# Oppgave 10

def f(x):
    return 2*x + 1


def apply_to_even(L):
    result = L
    for i in range(len(L)-1, -1, -1):
        if result[i] % 2 == 0:
            result[i] = f(result[i])
        else:
            del result[i]

    return result

L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result = apply_to_even(L)
print(result, id(result), id(L))
```

Python

```
[5, 9, 13, 17, 21] 1379337930496 1379337930496
```