Advanced Management of Data
Exercise 2 Topic 2:
Extensions of SQL

More parameters

- Task: write a new function `swap()`, that takes two integer parameters and returns them in reverse order
- hint: you also need two output parameters
- another hint: the keyword for output parameters is `OUT`

More parameters

```
CREATE OR REPLACE FUNCTION swap(INT, INT, OUT INT, OUT INT) AS
'SELECT $2, $1' LANGUAGE sql;
```

- the function body is just a string an can be escaped by ' or the $$-construct
- in SQL this is quite easy, but we also can do this with PL/pgSQL

```
CREATE OR REPLACE FUNCTION swap(INT, INT, OUT INT, OUT INT) AS $body$
  BEGIN
    $3 := $2; -- PL/pgSQL lets you assign like Oracle's PL/SQL
    $4 = $1;  -- but you can also use the standard SQL/PSM assignment
  END;
$body$ LANGUAGE plpgsql;
```

More types

```
SELECT * FROM swap(1, 2);
```

- swapping two integers is nice, but what about double precision floating-point numbers?
- Task: create a new function that is able to swap these
- hint: the type would be `DOUBLE PRECISION` or `FLOAT8`
- for more types see https://www.postgresql.org/docs/current/static/datatype.html

And more types

- as the next task will clearly be the same with some other type, let's just do it right now

```
CREATE OR REPLACE FUNCTION swap(IN ANYELEMENT, input2 ANYELEMENT, OUT
output1 ANYELEMENT, OUT ANYELEMENT) AS $$
  BEGIN
    output1 := input2; -- could still be written as $3 = $2
    $4 = $1;
  END;
$$ LANGUAGE plpgsql;
```

- parameters can be named for better readability of the code
- the keyword `IN` can be omitted as this is the default

Polymorphic functions

- Attention: in polymorphic functions all occurrences of `ANYELEMENT` must use the same type for a specific call (e.g. you can't mix integers and floating points)
- if your server doesn't detect the used type (this is always the case for string literals), you can give it a hint (otherwise the type "unknown" would be detected, which is not a valid type)

```
SELECT * FROM swap('test'::TEXT, 'string');
```

- obviously you just need to give the hint once, as they both must be the same type

Declarations

- using input and output parameters and literals for calculations is not always enough and you need local variables
- local variables have to be declared before they can be used
- Task: write a new function `twentyone()` which
  - declares two variables “seventeen” and “four”
  - assigns them the values “17” and “4”
  - return the sum of them (should be “21”)

Declarations
Blackjack

```
CREATE OR REPLACE FUNCTION twentyone() RETURNS INTEGER AS $$
  DECLARE
    seventeen INTEGER;
    four CONSTANT INT = 4; -- variables can be initialized on declaration
  BEGIN
    seventeen = 17;        -- … or later at usage
    RETURN seventeen + four;
  END;
$$ LANGUAGE plpgsql;
```

Swapping IN and OUT
Less parameter

- there are not only `IN` and `OUT` parameter, but also `INOUT` parameter
- Task: rewrite the `swap()` function to use only two parameters of type `INOUT`

Swapping IN and OUT
Less parameter

```
CREATE OR REPLACE FUNCTION swap(INOUT ANYELEMENT, INOUT ANYELEMENT) AS
'SELECT $2, $1' LANGUAGE sql;
```

- as before, the SQL function is quite easy and even shorter, but again we can do this with PL/pgSQL

```
CREATE OR REPLACE FUNCTION swap(INOUT ANYELEMENT, INOUT ANYELEMENT) AS $$
  DECLARE
    tmp $1%type; -- its not possible to declare it as ANYELEMENT
                 -- so make it the same type like the first parameter
  BEGIN
    tmp = $1;
    $1 = $2;
    $2 = tmp;
  END;
$$ LANGUAGE plpgsql;
```

Something more complex
Combine it all

- now it's time to combine all functions, we have written until now
- Task: write a new function that
  - takes a name as input parameter
  - assigns the return value of the function call to `twentyone()` to a local variable
  - assigns the value "42" to another variable
  - swaps the return values of the function calls to `hello()` and `hello(`*`name`*`)` and returns its return value
  - also returns the return value of a function call to `swap()` with the two local variables
- hint: it should return something like "Hello, *name*!", "Hello, World!", 42, 21

Something more complex
Combined it all

```
CREATE OR REPLACE FUNCTION combination(name VARCHAR, OUT str1 VARCHAR,
OUT str2 VARCHAR, OUT int1 INTEGER, OUT int2 INTEGER) AS $$
  DECLARE
    twenty_one CONSTANT INTEGER = twentyone();
    forty_two CONSTANT INTEGER = 42;
  BEGIN
    -- one could also just declare the first variable and then assign the
    -- value at this point like SELECT * INTO twenty_one FROM twentyone()
    SELECT * INTO str1, str2 FROM swap (hello(), hello(name));
    SELECT * INTO int1, int2 FROM swap (twenty_one, forty_two);
  END;
$$ LANGUAGE plpgsql;
```