Parentheses and Brackets

Math grouping with round ( )

```
prod = (a + b) * c
frac = L / (L + K_D)
```

Function calls with round ( )

```
print('Hello')
my_function(input_1, input_2)
```

Indexing with square [ ]

```
df['conc'] # 'conc'-column.
x_data[0] # First element of list/array.
```

Variables

Assigning a variable

```
my_var = 1000
my_other_var = 42 * my_var
```

Adding a column to a `DataFrame`

```
my_array = np.array([1, 2, 3, 4])
df['my_array_column'] = my_array
df['col_name'] = df['another_col'] * 2
```

Mathematical operations

Basic operations

```
1 + 1   # Sum
1 - 1   # Subtract
2 * 2   # Multiply
2 / 2   # Divide
2 ** 2  # Exponent
```

Math functions

```
np.exp(42)   # Exponential function.
np.log(42)   # Natural logarithm.
np.log10(42) # Base 10 logarithm.
np.sqrt(42)  # Square root.
```

Elementwise operations

```
arr_1 = np.array([1, 2, 3])
arr_2 = np.array([2, 3, 4])

arr_1 * arr_2 # ---> [2, 6, 12]
df['c1'] + df['c2'] # Sum elementwise
```

Basic Plotting

Create a figure

```
fig, ax = plt.subplots()
```

Line plot with label

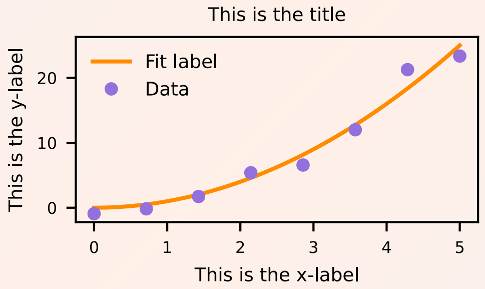
```
ax.plot(x_fit, y_fit, label='Fit Label')
```

Scatter plot with label.

```
ax.plot(df['x'], df['y'], 'o', label='Data')
```

Customization options

```
ax.legend() # Show legend with graph labels.
ax.set_xlabel('My x-label') # x-axis label.
ax.set_ylabel('My y-label') # y-axis label.
ax.set_title('My title') # The title.
ax.set_xscale('log') # Logarithmic x-axis.
```



Function Definition

Definition ends with colon - function body indented by 4 spaces or 1 tab.

```
def my_function(input_1, input_2):    # 1
    result = input_1 + input_2        # 2
    return result                     # 3
```

1: Function name and comma-seperated inputs, always end with colon.
2: Any number of lines of function logic.
3: `return`-statement specifies the function output.

For example,

```
def simple_binding(L, K_D):
    return L / (L + K_D)
```

Good idea to test with simple values

$$L = 1, K_D = 1 \rightarrow \frac{1}{1+1} = \frac{1}{2}$$

```
test_result = simple_binding(1, 1)
print(test_result) # Should give 0.5
```

Functions can be called with variables as arguments and return value assigned to a variable.

```
output = binding(df['L'], K_D_fit)
```

Curve Fitting

The first decision is always to choose an appropriate biochemical model

Making a fit

```
fit_params, trash = curve_fit(model_func,
                              x_data,
                              y_data,
                              initial_guess)
```

Extracting parameters

```
# Single parameter, e.g. simple_binding
K_D_fit = fit_params[0]
# Two parameters, e.g. Michaelis Menten
Vmax_fit = fit_params[0]
KM_fit = fit_params[1]
# Or generally
p1, p2, p3 = fit_params
```

Evaluating a fit

```
# Make linearly space points
x_fit = np.linspace(xmin, xmax, n_points)
# Calc. model with 1 parameter
y_fit = model_func(x_fit, param1)
# Calculate model with 2 parameters
y_fit = model_func(x_fit, p1, p2)
```

Calculating residuals

```
y_model =  model_func(x_data, param1)
residuals = y_data - y_model
```

DataFrame Operations

```
rate = df['rate']                 # Indexing
max_rate = df['rate'].max()       # Minimum
min_rate = df['rate'].min()       # Maximum
mean_rate = df['rate'].mean()     # Mean
rate0 = df['rate'][0]             # Index twice
```

By: Mads-Peter V. Christiansen