MATH5306M Introduction to Programming
2025/26

Assignment 3

In addition to their ability to quickly and accurately compute numerical calculations, by encoding appropriate rules and procedures, computers can be used to perform mathematical procedures in an ***analytic*** form – that is, through manipulation of abstract mathematical ***terms*** rather than just numerical *quantities* – examples include the Maple and Mathematica mathematics engiges, the latter of which powers the online Wolfram Alpha system. The final assignment consists of a guided project in which you will develop a programme capable of performing basic differentiation.

You should submit a single `.py` file containing functions performing the tasks detailed below. You should also submit a text file (in any readable format) describing your process in approaching these tasks – you should now be familiar with the approximate level of detail expected, but I won't impose any restriction on the length of this document – submit whatever you think expresses the relevant processes and decisions you have made in completing the assignment.

Submit by 12 midday, Monday 12th January 2025.

Part 1

Task:

Write a function which takes a string as input, and checks whether any brackets occurring in the string appear as correctly matched pairs of one opening bracket `(` and one closing bracket `)`. That is, the brackets in the following four strings are all correctly matched: `"()"`; `"()()"`; `"(())"` `"(()())"`; . However, in the following strings, the brackets are not correctly matched: `"("`; `")("`; `"())"`; `"())(()`". Your programme should output `True` if all brackets in the string are correctly matched and `False` otherwise. Note that, since we are mathematicians, a string in which no brackets occur whatsoever is a string in which all brackets appear as correctly matched pairs. We are only considering ordinary round brackets, not any other kind of bracket.

Hint: If a string contains mismatched brackets, we must have either opened a pair of brackets which we haven't subsequently closed, or (looking from the start of the string to the end) tried to close more pairs of brackets than we have opened.

Test inputs:

Input	Expected output
`"()"`	`True`
`"a"`	`True`
`"(b())"`	`True`
`"("`	`False`
`"()a)"`	`False`
`"|)"`	`False`
`"(]"`	`False`
`"([)"`	`True`
`"[["`	`True`
`"(a(bb)ccc)dddd"`	`True`
`"a(bb)()ccc"`	`True`

Part 2:

Task:

Write a function which takes a string as input and, in the case that any brackets present in the string occur in correctly-matched pairs, returns the *contents* of the brackets (i.e. all of the characters contained in the string that are not the `(` character or the `)` character), in an appropriate structure of your choosing. In the case the input string contains mismatched brackets your function should print an error message and `return None`.

Part 3:

Mathematical expressions and well-formed terms

A mathematical expression consists of various *terms* connected by mathematical *operators*, for example the expression $2x + y$ has terms $2x$ and y and the operator $+$. Of course the term $2x$ itself is actually an abbreviated shorthand for $2 \times x$, which contains the terms 2 and x and the operator $\times$.

We will define a well-formed term in the following way:

Rule	Examples
Any single letter (which we will interpret as a variable, in the mathematical sense) is a well-formed term	`x` is a well-formed term; `a` is a well-formed term;
Any natural number is a well-formed term	`3` is a well-formed term; `52837` is a well-formed term;
If s and t are well-formed terms, then:	
(s+t) is a well-formed term	`(x+3)` is a well-formed term, where s is the term `x` and t is the term `3`; `(52837+(x+3))` is a well-formed term, where s is the term `52837` and t is the term `(x+3)`;
(s-t) is a well-formed term	`(y-1)` is a well-formed term, where s is the term `y` and t is the term `1`; `((x-3)-y)` is a well-formed term, where s is the term `(x-3)` and t is the term `y`;
(s*t) is a well-formed term	`(2*(y-1))` is a well-formed term, where s is the term `2` and t is the term `(y-1)`; `((x-(2*x))*y)` is a well-formed term, where s is the term `(x-(2*x))` and t is the term `y`;
(s/t) is a well-formed term	`(a/b)` is a well-formed term, where s is the term `a` and t is the term `b`; `(1/(x+1))` is a well-formed term, where s is the term `1` and t is the term `(x+1)`;

Rule	Examples
If s is a well-formed term and n is a natural number greater than or equal to 1, then:	
`(s^n)` is a well-formed term	`(x^2)` is a well-formed term; `((y*(x-(2*x)))^52837)` is a well-formed term.

Note that we can apply our formation rules in any order. For example, `(((x^2)*(x/(y^3)))+1)` is a well-formed term.

Task:

Write a function which takes a string as input and determines whether its contents are a well-formed term.

Hint: Note that we have essentially ***three*** kinds of well-formed terms: single letter variables, natural numbers, or terms consisting of an opening/closing pair of brackets and an operator (i.e. one of `+`, `-`, `*`, `/`, `^`), to the left and right of which is a further well-formed term (– i.e. either a single letter variable, a natural number, or a term enclosed in a pair of brackets with an associated operator). Note also that every individual operator corresponds to a specific opening left bracket and closing right bracket. For example, the `+` operator in the last example corresponds to the outside brackets: `(((x^2)*(x/(y^3)))+1)`, and the `*` operator corresponds to the brackets indicated in blue: `(((x^2)*(x/(y^3)))+1)`

Part 4:

We are going to extend our definition of a well-formed term to allow us to consider further mathematical functions, by including the following rules: (remember, a well-formed term is just a string of text, it doesn't include any actual mathematical functionality)

Rule	Examples
If t is a well-formed term, then:	
`exp(t)` is a well-formed term	`exp(x)` is a well-formed term;
`log(t)` is a well-formed term	`log((a+b))` is a well-formed term;
`sin(t)` is a well-formed term	`(sin(1)/y)` is a well-formed term;
`cos(t)` is a well-formed term	`cos(exp((x/2)))` is a well-formed term;

Now, we have four distinct kinds of terms. From before, we have terms which are either a single letter variable, a natural number, or a term enclosed in a pair of brackets corresponding to an operator. We now have a fourth kind of term, consisting of a function name followed by a corresponding pair of brackets.

Symbolic Differentiation

We will have learned at school how to differentiate mathematical functions with respect to a single variable. Let us consider differentiation with respect to the variable x. This means that any other single-letter variable just represents a constant coefficient. We have the following well-known derivatives:

Function $\left(f(x)\right)$	Derivative with respect to x $\left(\frac{df}{dx}\right)$
Any constant (including natural numbers and variables other than x)	0
x	1
x^n where n is a natural number greater than 1	$n \times x^{n-1}$
$\exp(x)$	$\exp(x)$
$\log(x)$ (Natural logarithm)	$\frac{1}{x}$
$\sin(x)$ (angle x in radians)	$\cos(x)$
$\cos(x)$ (angle x in radians)	$-\sin(x)$

Furthermore, given functions f and g, we have the following well-known rules for the derivative of various combinations of f and g, in terms of their derivatives:

Function	Derivative with respect to x
$f+g$	$\frac{df}{dx} + \frac{dg}{dx}$
$f \cdot g$	$f \cdot \frac{dg}{dx} + g \cdot \frac{df}{dx}$ (Product rule)
$f(g(x))$	$\frac{dg}{dx} \times \frac{df}{dx}(g(x))$ (Chain rule)
f/g	$\frac{g \cdot \frac{df}{dx} - f \cdot \frac{dg}{dx}}{g^2}$ (Quotient rule)

Task:

Use these rules and derivatives to write a function which takes a mathematical expression (as a `Python` string) as input, and returns (as a string) an expression giving the derivative with respect to x of the input expression. Here you should interpret `*` as multiplication, `^` as raising to a power (note we are not using the Python syntax for raising to a power), `exp()` as the exponential function, `log()` as the natural logarithm, and `sin()` and `cos()` as being the sine and cosine of an angle in radians. You should implement these rules directly in your programme, that is, you must not use any pre-written modules etc. which manipulate mathematical expressions. Your function should perform correctly on all well-formed terms as described above; your function should also accept (that is, perform correctly on) as input any expression that it may return as output.

Hint: Note that per the above rules for differentiation, given a term `x^2` as input (i.e. an expression representing the mathematical function x^2), its derivative is defined as "$2 \times x^1$", however the derivative of the term `x^1` is not covered by the same definition.

Task:

Write a programme which takes a mathematical expression as input from the user and prints its derivative. You may wish to consider further aspects of the user interface in terms of suitable/meaningful input and output. Once again, you must not use any pre-written modules etc. which manipulate mathematical expressions.