NATIONAL UNIVERSITY OF SINGAPORE
Department of Computer Science, School of Computing
IT5001—Software Development Fundamentals
Academic Year 2023/2024, Semester 2
PROBLEM SET 1

EXPRESSIONS & CALLABLE UNITS

These exercises are optional and ungraded, and for your own practice only. Contact `yongqi@nus.edu.sg` for queries.

SIMPLE EXERCISES

Question 1 (Expression Evaluation). Without using IDLE, evaluate the results of the following expressions:

1. `3 * 4 + 5`
2. `3 + 4 * 5`
3. `5 ** 3 % 4`
4. `97 / 4`
5. `97 // 4`
6. `((1 + 2 - 3) ** (4 % 5 // 6)) * 7 * 'a'`
7. `f'x is {4 - 2:05d}'`
8. `'Your burger ' + f'costs ${10:.2f}'`
9. `'0012' in f'Your queue number is {12:010d}'`
10. `'\'abc\'' in 'zabcdef'`
11. `3 + 2 == 4.0 + 1`
12. `'abcdefgh'[:-1] != 'abcdefg'`
13. `4 >= 4`
14. `0 == 0 == 0`
15. `1 <= 4 < 7`
16. `1 == '1'`
17. `123 + 456`
18. `'123' + '456'`
19. `'123' + 456`
20. `'1234' < '1234567'[:3]`
21. `'abcdefgh'[2:9]`
22. `'abcdefgh'[100:-20:-2]`

```
'abcdefgh'[3:2]
'Abc' > 'abc'
'bca' < 'bca0'.upper()
ord('back'[1])
chr(65)
True and True
True and False
False or True
False or False
1 and 0
1 or 2
not ''
not 'abc'
not not True
not 0
not 9999
'abc'[3]
1 > 'a'
'abc'[]
'abc'[:]
True and (False or True)
True + 1
False * 5
0 + (not 1)
int(1.01)
int(1.99)
int(-2.99)
bool('False')
type('123')
1 or (1 / 0)
'' and (1 / 0)
```

Question 2. What is the output of the following program:

```
x = 3
y = 4
z = 5
print(x * y + z)
```

Question 3. Consider the following code snippet:

```
x = 5 > 3
print(x)
x = 2
print(x)
```

What does this tell you about variables?

Question 4. Consider the following code snippet:

```
m += 1
print(m)
```

Why does this lead to an error?

SHORT PROGRAMMING QUESTIONS

Question 5. Write a program that reads in 2 nonempty strings and prints `True` if their last (rightmost) character is the same and `False` otherwise.

Question 6. Write a program that reads in 3 numbers and prints the smallest integer greater than or equal to their average (mean) value.

Question 7. Write a program that reads in a number and prints 1 if it is truthy and 0 if it is falsey.

Question 8. Write a program that reads in n and d from the user and prints the number after removing the d rightmost digits of n. For example, if the user enters 123456 as n and 3 as d, then your program should print 123. Assume that the user enters valid positive integers n and d where $\log_{10} n \geq d$.

Question 9. Write a program that reads in the current time in 24h format, t, and some number of hours n, then print the time n hours after t, in 24h format. For example, if the user enters 0159 as t and 50 as n, then your program should print 0359. Assume that the user enters a valid t and nonnegative integer n.

Question 10. Write a program that reads in l, and draws a pentagon (5-sided polygon) where the longest length across any 2 vertices of it is l.

In Figure 1, the horizontal line depicts the longest length across any 2 vertices of the pentagon. Your pentagon should be drawn such that drawing such a line will result in one of length l.

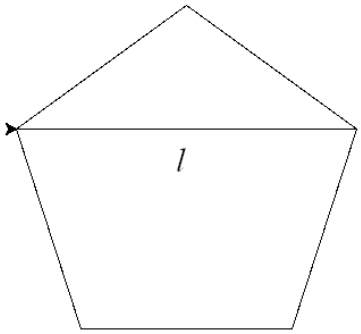
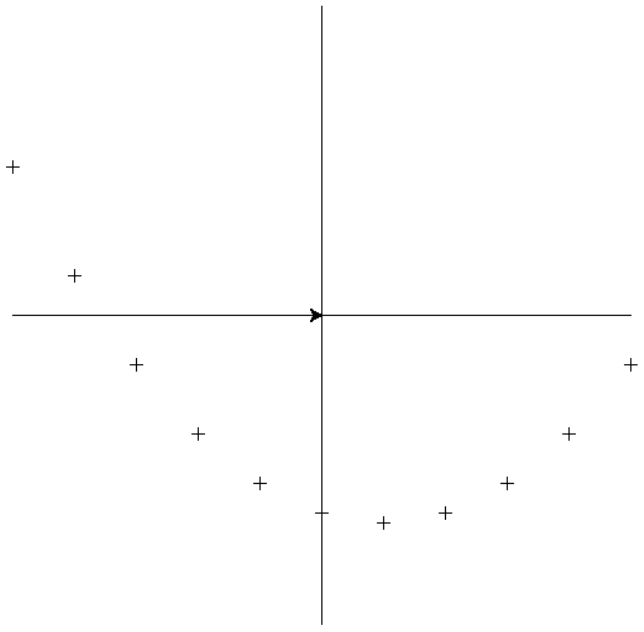


Figure 1. A pentagon where the longest length across any 2 vertices is l.

LONG PROGRAMMING QUESTION

We are going ask our friendly turtle to plot a graph of any univariate polynomial with degree up to 3. A univariate polynomial is an expression consisting of one variable (also called an indeterminate) and coefficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponentiation of variables. The degree of said polynomial is the maximum degree of exponentiation of the variable in any of its terms. For example, the polynomial $f(x) = 5x^4 - 2x^3 + 48$ is a polynomial with degree 4. Our program is going to plot graphs for polynomials with degree up to 3.



Example output of the program plotting $f(x) = 2x^2 - 4x - 40$.

Drawing Axes

The first thing we would want to do is to draw some axes. Write a function `plot_axes(width, height)` which plots the axes where the origin is the turtle's original position.

Plotting Points

The next thing we should be able to do is to plot a point. Write a function `plot_point(x, y)` that plots a point on the graph at coorindates (x, y). The mini cross showing the point should be 10-by-10 pixels.

Hint: to allow the turtle to travel without drawing anything, you can use the `penup()` function. For the turtle to continue drawing, use `pendown()`.

Drawing any cross

Notice that in both parts 1 and 2 you are drawing crosses? What we should have done is to write some general business logic that can draw any cross of any width and height, and at any location, and let `plot_axes` and `plot_point` make use of that function.

Write a new function `draw_cross(x, y, width, height)` that draws a cross of `width` and `height` at coordinates (x, y). Then, rewrite `plot_axes` and `plot_point` such that it only relies on your new `draw_cross` function.

Note: this process is called refactoring—which is the process of restructuring existing computer code without changing its external behavior. Refactoring is intended to improve the design, structure, and/or implementation of the software, while preserving its functionality. *Source: Wikipedia.*

Solving Polynomials

The next thing we should be able to do is, given the coefficients and x value of a polynomial, return the corresponding y value.

Write a function `eval_cubic_eqn(x, c0, c1, c2, c3)` that returns $c_3x^3 + c_2x^2 + c_1x + c_0$.

Plotting the Graph

Now it is time to put everything together. Write a function `plot_poly(c0, c1, c2, c3)` that plots a graph of the polynomial of 500-by-500 pixels. The x-values should range from -5 to 5 and be in intervals of 1. However, to make use of the full x-axis, scale the x-position by 50 pixels (giving us a range of -250 to 250 pixels) when plotting. c_1, c_2, c_3 should be optional to allow the function caller to plot polynomials with lower degrees.

Sometimes the y-values may get so big that it far exceeds the size of the plot. To fix that you may allow an optional `y_scale` parameter in your function that scales the y-values to something more reasonable.

Note how cumbersome it is to plot so many points by copy and pasting the same calls to `plot_point`. In Week 2 you will learn how to repeat a bunch of statements any number of times easily!

SOLUTIONS

Question 1. Finding out the solutions to these is quite simple—just write the expressions in IDLE to see the results after evaluating them.

Here are the explanations for each expression.

(1.1) Answer: 17. The multiplication is done first.

(1.2) Answer: 23. As above.

(1.3) Answer: 1. The exponentiation is done first (giving us 125), followed by the modulo.

(1.4) Answer: 24.25. This is normal division.

(1.5) Answer: 24. This is floor division—therefore we simply take the floor of 24.25 (which is 24).

(1.6) Answer: `'aaaaaaa'`. The parenthesized expressions are evaluated first, giving us `(0 ** 0) * 7 * 'a'`. In Python, $0^0 = 1$. This gives us `1 * 7 * 'a'`, and the result of this is obvious.

(1.7) Answer: `'x is 00002'`. We first determine what is `{4 - 2:05d}`, which is essentially `{2:05d}`. This says that we want to format `2` as a zero-padded, 5-digit decimal number (an integer, basically), giving us the string `'00002'`.

(1.8) Answer: `'Your burger costs $10.00'`. `{10:.2f}` says we want to format `10` as a two-decimal place floating point number, which gives us `'10.00'`.

(1.9) Answer: **`True`**. We first evaluate the string `f'Your queue number is \{12:010d\}'`, which turns out to be `'Your queue number is 0000000012'` (we formatted `12` as a zero-padded, 10-digit decimal number). Quite clearly, the string `'0012'` is a substring of `'Your queue number is 0000000012'`, explaining the result.

(1.10) Answer: **`False`**. The string `'\'abc\''` actually looks like `'abc'` (try printing it out), but the string itself includes the quotation marks. Therefore it is not a substring of `'zabcdef'`.

(1.11) Answer: **`True`**. Despite one being an integer and the other being a floating point number, equality comparisons between integers and floating point numbers really only care about the equality of the mathematical values which they represent.

(1.12) Answer: **`False`**. `'abcdefgh'[:-1]` gives us `'abcdefg'` (the slice starts from the beginning of the string, ending and not including the last character in the string). `'abcdefg'` is indeed equal to `'abcdefg'`.

(1.13) Answer: **`True`**. Trivial.

(1.14) Answer: **`True`**. `0 == 0 == 0` can be re-written as `0 == 0 and 0 == 0`.

(1.15) Answer: `True`. Can be re-written as `1 <= 4 and 4 < 7`

(1.16) Answer: `False`. Even though they look similar, an integer will never be equal to a string, because they are of different data types.

(1.17) Answer: 579. Comes as no surprise.

(1.18) Answer: `'123456'`. Unlike the above, the + operator on two strings denotes string concatenation.

(1.19) Answer: You get a type error. + is not defined on a string and an integer.

(1.20) Answer: `False`. `'1234567'[:3]` gives us `'123'`. By the rules of lexicographical ordering, the result is as expected.

(1.21) Answer: `'cdefgh'`. We start at index 2 (the c), and end (without including) at index 9 (beyond the h).

(1.22) Answer: `'hfdb'`. Slicing using indices beyond the length of the string is legal. i.e. starting from index 100 would just default to starting at `'h'`.

(1.23) Answer: `''`. No possible slice can be created that starts at index 3 and ends (exclusive) at index 2. When this happens, the empty string is produced.

(1.24) Answer: `False`. `'A'` has a smaller unicode code point than `'a'`.

(1.25) Answer: `False`. The method will be invoked first, before the comparison is being made. The expression therefore reduces to `'bca' < 'BCA0'`.

(1.26) Answer: 97. `'back'[1]` gives `'a'`.

(1.27) Answer: `'A'`. 65 is the unicode code point of `'A'`.

(1.28) Answer: `True`. Both values beside the `and` operator are `True`.

(1.29) Answer: `False`. One of the values beside the `and` operator is `False`.

(1.30) Answer: `True`. One of the values beside the `or` operator is `True`.

(1.31) Answer: `False`. Both values beside the `or` operator are `False`.

(1.32) Answer: `0`. When the value on the left of the `and` operator is truthy, the expression evaluates to the value on the right side of the `and` operator.

(1.33) Answer: `1`. When the value on the left of the `or` operator is truthy, the expression evaluates to that value.

(1.34) Answer: `True`. The empty string is falsey. This expression is commonly used to determine if a string is empty.

(1.35) Answer: `False`. A non-empty string is truthy.

(1.36) Answer: `True`. Trivial.

(1.37) Answer: `True`. 0 is False. This expression is commonly used to determine if a number is equal to 0.

(1.38) Answer: `False`. A non-zero number is truthy.

(1.39) Answer: You get an IndexError. That is because no element in the string has index 3.

(1.40) Answer: You get a TypeError. Inequalities between integers and strings are undefined.

(1.41) Answer: You get a SyntaxError. You cannot omit the index while indexing an element.

(1.42) Answer: `'abc'`. This essentially copies the string. Unlike above, you can omit indices when slicing strings (we know we're slicing because of :).

(1.43) Answer: `True`. We evaluate the parenthesized expression first.

(1.44) Answer: 2. `True` is equal to 1.

(1.45) Answer: 0. `False` is equal to 0.

(1.46) Answer: 0. Evaluating the parenthesized expression first, `not 1` gives us `False`.

(1.47) Answer: 1. Converting a floating point number into an integer truncates the digits after the decimal point.

(1.48) Answer: 1. As above.

(1.49) Answer: -2. As above.

(1.50) Answer: `True`. Don't be fooled—the nonempty string is truthy.

(1.51) Answer: It should show `<class 'str'>`, denoting that it is of the string type.

(1.52) Answer: 1. In the expression `1 or (1 / 0)`, notice that since `1` is already a truthy value, it is not even necessary to evaluate `(1 / 0)` since regardless of whether it is truthy or falsey, the entire must definitely be truthy.

(1.53) Answer: `''`. Similar reasoning to 1.52, except that the operator here is `and` and that `''` is falsey.

Question 2. The assignment statements store 3 in x, 4 in y and 5 in z. When we are printing `x * y + z`, we are really printing the result of evaluating `3 * 4 + 5`.

Question 3. It tells us that variables can store any data type throughout its lifetime.

Question 4. It leads to an error because `m += 1` is essentially `m = m + 1`. To evaluate the assignment statement, we need to determine the object on the right hand side. However, we cannot evaluate what `m + 1` is because at that point, `m` has not been defined yet.

Question 5. Your program may look like this:

```
x = input()
y = input()
print(x[-1] == y[-1])
```

Alternatively, we may use positive indexing (incrementing by 1 from left to right):
`x[len(x) - 1] == y[len(y) - 1]`. Here, the function `len` gives you the number of characters in the input string. Which approach is better?

Question 6. Your program may look like this:

```
from math import ceil

x = float(input())
y = float(input())
z = float(input())
print(ceil((x + y + z) / 3))
```

Note that the inputs may not be given as whole numbers. The `math` library also contains the `floor` function, which gives the largest integer less than or equal to the input number.

Question 7. Your program may look like this:

```
print(int(bool(float(input()))))
```

In Python, **`True`** is represented as the value 1, and **`False`** is represented as the value 0.

Question 8. Your program may look like this:

```
n = int(input())
d = int(input())
print(n // 10 ** d)
```

Firstly, read n and d from the user and convert them into integers. Remember, to remove the d rightmost digits of n we would do $\lfloor \frac{n}{10^d} \rfloor$, for example, removing the 2 rightmost digits of 12345 we would do $\lfloor \frac{12345}{100} \rfloor$, or in Python, `12345 // 100`. We just needed to generalize this logic for any valid n and d.

Question 9. Your program may look like this:

```
t = int(input())
n = int(input())
hour_of_time = t // 100
minute_of_time = t % 100
new_hour = (hour_of_time + n) % 24
new_time = new_hour * 100 + minute_of_time
print(f'{new_time:04d}')
```

Firstly, read t and n as integers. Then, obtain the 'hour' of the time by removing the two rightmost digits of the number (i.e. you should get 11 from 1136). Then, obtain the 'minutes' of the time by obtaining the two rightmost digits of the number (i.e. you should get 36 from 1136), and this can be done by a simple modulo by 100.

Once you have the current hour, add n to it to obtain the new hour (for example, if the current hour is 11 and n is 20, you should get 31). Importantly, this may give you a very big number. To 'wrap' it around a 24h clock, you need to do a modulo by 24 (for example, we can wrap 31 around a 24h clock by doing 31 mod 24 which would give 7. Indeed, 20 hours after 11am is 7am).

The new time can be obtained by multiplying the new hour by 100 (obtaining 700 from 7) then adding it to the minute of time (700 + 36 would be 736. Indeed, 20 hours after 11.36am is 7.36am). To print the new time out as a 4 digit number padded by zeroes, we use the format specifier `04d`.

Question 10. Your program may look something like this:

```
from math import sin, radians
from turtle import forward, left
l = int(input())

# Obtain the side length
side_length = l * sin(radians(36)) / sin(radians(108))

# Draw the pentagon
forward(side_length)
left(72)
forward(side_length)
left(72)
forward(side_length)
```

```
left(72)
forward(side_length)
left(72)
forward(side_length)
left(72)
```

This involves a bit of geometry. We know the angles at the vertices of the pentagon is $(180\times3)\div5 = 108$. This means that the triangle created by drawing the horizontal line in the pentagon has angles 108, 36, 36. Therefore, we obtain the following relation between l and the side length s:

$$\frac{s}{\sin(36)} = \frac{l}{\sin(108)}$$

Therefore, the side length is simply $l \times \sin(36) \div \sin(108)$. Of course, the `sin` function in the `math` library takes input angles in radians, not degrees, so we need to convert 36 and 108 into radians first, which we can do with the `radians` function.

Finally, to draw the pentagon, we just need to draw the five edges. At every edge we have to turn left by $180 - 108 = 72$ degrees.

Long Programming Question. Your program may look like this:

```
from turtle import *
def draw_cross(x, y, width, height):
    # go to (x, y)
    penup()
    goto(x, y)
    pendown()
    # draw vertical line
    forward(width // 2)
    backward(width)
    forward(width // 2)
    left(90)
    forward(height // 2)
    backward(height)
    forward(height // 2)
    right(90)
    # bring back to (0, 0) just in case
    penup()
    goto(0, 0)
    pendown()

def plot_axes(width, height):
    # make use of draw_cross function!
    draw_cross(0, 0, width, height)
```

```
24
25 def plot_point(x, y):
26     # make use of draw_cross function!
27     draw_cross(x, y, 10, 10)
28
29 def eval_cubic_eqn(x, c0, c1, c2, c3):
30     # simple math
31     return c3 * x ** 3 + c2 * x ** 2 + c1 * x + c0
32
33 # use default parameter values to keep higher coefficients optional!
34 def plot_poly(c0, c1 = 0, c2 = 0, c3 = 0, y_scale = 1):
35     # plot the axes
36     plot_axes(500, 500)
37     # plot each point
38     # in Week 2 we will learn how to repeat a bunch of statements more
39     # easily
40     plot_point(5 * 50, eval_cubic_eqn(5, c0, c1, c2, c3) * y_scale)
41     plot_point(4 * 50, eval_cubic_eqn(4, c0, c1, c2, c3) * y_scale)
42     plot_point(3 * 50, eval_cubic_eqn(3, c0, c1, c2, c3) * y_scale)
43     plot_point(2 * 50, eval_cubic_eqn(2, c0, c1, c2, c3) * y_scale)
44     plot_point(1 * 50, eval_cubic_eqn(1, c0, c1, c2, c3) * y_scale)
45     plot_point(0 * 50, eval_cubic_eqn(0, c0, c1, c2, c3) * y_scale)
46     plot_point(-1 * 50, eval_cubic_eqn(-1, c0, c1, c2, c3) * y_scale)
47     plot_point(-2 * 50, eval_cubic_eqn(-2, c0, c1, c2, c3) * y_scale)
48     plot_point(-3 * 50, eval_cubic_eqn(-3, c0, c1, c2, c3) * y_scale)
49     plot_point(-4 * 50, eval_cubic_eqn(-4, c0, c1, c2, c3) * y_scale)
50     plot_point(-5 * 50, eval_cubic_eqn(-5, c0, c1, c2, c3) * y_scale)
```

– End of Problem Set 1 –