

1. Getting Started

2. More Details About Pyth

3. Some Simple Programs

4. Some Simple Programs - Part II

5. Learning More - Documentation and Errors

6. Adding to Pyth

7. The Language Specification - Variables


8. The Language Specification - Control Flow

9. The Language Specification - Arithmetic

10. The Language Specification - Comparisons

11. The Language Specification - Sequences


The best way to build AI-powered apps

 **START BUILDING**

GenAI apps + MongoDB Atlas

You don't need a separate database to start building GenAI-powered apps.

Ad by EthicalAds



4. Some Simple Programs - Part II

Collatz Sequence

Another relatively simple programming problem for us to golf down is to generate Collatz sequences. The Collatz sequence consists of repeatedly applying the following procedure: If the current number is even, divide it by two. If the current number is odd, triple it and add one. Repeat until 1 is reached. Suppose we would like to print out the entire sequence, from the input to 1. Here is a very straightforward implementation, which illustrates many of Pyth's statements:

```
input: 3

===== 23 chars =====
WtQI%Q2=Qh*Q3).?=Q/Q2)Q
=====
assign('Q',Punsafe_eval(input()))
while tail(Q):
    if mod(Q,2):
        assign('Q',head(times(Q,3)))
    else:
        assign('Q',div(Q,2))
    imp_print(Q)
=====
10
5
16
8
4
2
1
```

Here, we use a while loop, `W`, with an if statement, `I`, and an else statement, `.?` in the body. The if and else adjust the value of the changing variable, `Q`, then the value is printed with the final `Q`. The loop condition is checked using `tQ`, which, since `t` is the decrement function, is nonzero and thus truthy whenever `Q!=1` as desired. `%Q2`, corresponding to `Q%2` in most languages, is truthy whenever `Q` is odd.

As an improvement, we can use `?`, Pyth's `if ... then ... else ...` operator. Like C, but unlike Python, the conditional goes in front, then the truthy branch, then the falsy branch. This allows us to get rid of the statement overhead and the repetition of `=Q`:

```
===== 17 chars =====
WtQ=Q?%Q2h*Q3/Q2Q
=====
assign('Q',eval(input()))
while tail(Q):
    assign('Q',(head(times(Q,3)) if mod(Q,2) else div(Q,2)))
    imp_print(Q)
=====
10
5
16
8
4
2
1
```

That saved 5 characters, or 23%. However, we can still do better. Pyth's assignment and lookup functions, `X` and `@` respectively, have a feature which is very handy in exactly this situation. Assignment and lookup wrap around when called on sequences, which means that if we lookup the `Qth` location of a sequence consisting of the two possible Collatz sequence successors, the proper one will be selected. Fortunately, the perfect function exists, `.`, which constructs a 2-tuple of its 2 arguments. Putting these together gives us:

```
===== 16 chars =====
WtQ=Q@./Q2h*3QQQ
=====
Q=copy(literal_eval(input()))
while tail(Q):
    Q=copy(lookup((div(Q,2),head(times(3,Q))),Q))
    Pprint("\n",Q)
=====
10
5
16
8
4
2
1
```

That's as short as it can be, as far as I know.