1.3-built-in-functions-and-libraries-solutions

April 19, 2021

---

1 Built-in functions and Libraries

EXERCISES *1. Run the following python code and think what each of the print statements in the code below will print.*

```
easy_string = "abc"
print(max(easy_string))
rich = "gold"
poor = "tin"
print(max(rich, poor))
print(max(len(rich), len(poor)))
```

SOLUTION

[1]:
```
easy_string = "abc"
print(max(easy_string)) # max() returns the highest alphabetical character in a␣
↪string.
```

```
c
```

[2]:
```
rich = "gold"
poor = "tin"
print(max(rich, poor)) # It "orders" the words alphabetically and returns the␣
↪one that is at the bottom of the alphabetic list, it returns tin bacause t␣
↪is after g
```

```
tin
```

[3]:
```
print(max(len(rich), len(poor))) # return the max number of characters from␣
↪these two strings, aka 4 from "gold"
```

```
4
```

- *1b. Does* `max(len(rich), poor)` *run or produce an error message? If it runs, does its result make any sense?*

[4]:
```
max(len(rich), poor)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-bc82ad05177a> in <module>()
----> 1 max(len(rich), poor)

TypeError: '>' not supported between instances of 'str' and 'int'
```

It throws a TypeError. This turns into max(4, 'tin') and as we discussed earlier a string and integer cannot meaningfully be compared.

---

2. What function from the ***math*** *module can you use to calculate a square root without using* ***sqrt****?*

SOLUTION

1. Using help(math) we see that we've got pow(x,y) in addition to sqrt(x), so we could use pow(x, 0.5) to find a square root. The sqrt(x) function is arguably more readable than pow(x, 0.5) when implementing equations. Readability is a cornerstone of good programming, so it makes sense to provide a special function for this specific common case.

---

3. The following variable stores one of the longest word in the world.

```
longest_word = 'pneumonoultramicroscopicsilicovolcanoconiosis'
```

Suppose you want to select a random character from ***longest_word****:*

Which standard library module could help you?

- *3.2. Which function would you select from that module? Are there alternatives*

SOLUTION

The random module seems like it could help you.

The string has 45 characters, each having a positional index from 0 to 44. You could use random.randrange function (or the alias random.randint if you find that easier to remember) to get a random integer between 0 and 10, and then pick out the character at that position:

[8]:
```
from random import randrange
longest_word = 'pneumonoultramicroscopicsilicovolcanoconiosis'
print(len(longest_word))
random_index = randrange(len(longest_word))
print(longest_word[random_index])
```

```
45
i
```

Perhaps you found the random.sample function? It allows for slightly less typing:

[9]:
```
from random import sample

print(sample(longest_word, 1)[0])
```

```
n
```

---

4. *Fill in the blanks so that the program below prints 90.0.*

```
import math as m
angle = ____.degrees(____.pi / 2)
print(____)
```

SOLUTION

[13]:
```
import math as m
angle = m.degrees(m.pi / 2)
print(angle)
```

```
90.0
```

---

5. Rewrite the code above so that it uses import without as. Which form do you find easier to read?

[14]:
```
import math
angle = math.degrees(math.pi / 2)
print(angle)
```

```
90.0
```

SOLUTION

Since you just wrote the code and are familiar with it, you might actually find the first version easier to read. But when trying to read a huge piece of code written by someone else, or when getting back to your own huge piece of code after several months, non-abbreviated names are often easier, except where there are clear abbreviation conventions.
