01z_AppendixPythonBuiltins

May 17, 2022

1 Python Built-In Functions

Commands (“functions”) which we can use without importing anything…

- print()
- len()
- sum()
- input()
- conversion
  - str()
  - int()
  - float()
  - list()
- f””
  - string formatting

1.1 `print()`

`print()` outputs to the screen its input *arguments* separated by a space, and ending with a newline (a blank line).

[4]:
```python
print("Hello", "World")
print("-- K&R")
```

```
Hello World
-- K&R
```

We can change the separater used. Below we “configure” the print function using a *named* argument…

[5]:
```python
print("Hello", "World", sep=",") # named argument "keyword argument"
```

```
Hello,World
```

Using `end=` we can stop `print()` outputting a newline…

[6]:
```python
print("Hello", "World", end=" ")
print("-- K&R")
```

```
Hello World -- K&R
```

1.2 `print()` vs. `=`

`print()` sends text to the *screen*, not to *memory*. Anything you send is thrown away.

If you want to save a result, use a variable,

```
[46]: result = "LOAN APPLICATION APPROVED!"
```

And then later on you can print it (typically at the end of your program),

```
[47]: print(result)
```

```
LOAN APPLICATION APPROVED!
```

1.3 `len()`

`len()` counts the number of elements in data sets ("collections")...

```
[7]: prices = [1, 2, 5, 7, 8]

     len(prices)
```

```
[7]: 5
```

`len()` on text counts... the number of characters.

```
[8]: len("Michael")
```

```
[8]: 7
```

1.4 `sum()`

```
[9]: sum(prices)
```

```
[9]: 23
```

1.5 `input()`

Ask a user a question, and store the answer (as text):

```
[10]: age = input("What's your age?")
```

```
What's your age?31
```

```
[11]: age
```

```
[11]: '31'
```

1.5.1 Aside: the problems with text...

Operations applied to text are always *specialized* to text..

```
[16]: "Michael" + "Burgess"
```

```
[16]: 'MichaelBurgess'
```

```
[48]: 5 + 5
```

```
[48]: 10
```

Also, *

```
[49]: 10 * 2
```

```
[49]: 20
```

```
[12]: age * 2
```

```
[12]: '3131'
```

```
[13]: 'Ho' * 3
```

```
[13]: 'HoHoHo'
```

...the name of the specialization is called *polymorphism*.

1.6 Conversion Functions

Conversion functions are necessary because operations are specialized to the type of data they operate on. We have to convert to the right type before we run the operation.

```
[14]: age_numeric = float(age) #convert age to floating point number
```

```
[15]: age_numeric * 2
```

```
[15]: 62.0
```

```
[18]: 'Michael likes ' + str(5)
```

```
[18]: 'Michael likes 5'
```

```
[19]: [1, 2, 3] + [4, 5]
```

```
[19]: [1, 2, 3, 4, 5]
```

```
[20]: list("Michael")
```

```
[20]: ['M', 'i', 'c', 'h', 'a', 'e', 'l']
```

1.7 String Formatting

We format text for output using `print()`, but this always outputs text to the screen...

```
[22]: name = "Michael"
      city = "London"
      age = 31

      print(name, city, age)
```

```
Michael London 31
```

`print()` does two things, formats its input (gluing, converting to text) and also *outputting to the screen...*.

```
[23]: msg = name + ' ' + city + ' ' + str(age)
```

```
[24]: msg
```

```
[24]: 'Michael London 31'
```

String formatting using the `f` command allows us to *interpolate* (substitute) python code in our text blocks (strings) more easily than using lots of `+`s...

```
[28]: fmt = f"This person is called {name}, he lives in {city} and is {age} years old!
       ↪"
```

```
[29]: fmt
```

```
[29]: 'This person is called Michael, he lives in London and is 31 years old!'
```

```
[30]: print(fmt)
```

```
This person is called Michael, he lives in London and is 31 years old!
```

These strings are just text... they can be stored anywhere text can be...

```
[33]: ages = [31, 27]
```

```
[34]: msgs = [
          f"Micahel is {ages[0]}",
          f"Alice is {ages[1]}",
      ]
```

```
[35]: msgs
```

```
[35]: ['Micahel is 31', 'Alice is 27']
```

1.7.1 Aside

You an use complex expressions inside formatted strings,

```
[36]: f"a complicated expression is: { 2 ** 3 + 10 - 5} "
```

```
[36]: 'a complicated expression is: 13 '
```

```
[38]: people = {
          "names" : ["michal", "alice"],
          "ages"  : [31, 267]
      }
```

```
[43]: f"the first person is {people['names'][0]} and the last age is␣
       ↪{people['ages'][-1]}" # why are single quotes required here?
```

```
[43]: 'the first person is michal and the last age is 27'
```

```
[45]: people["names"][0] # but not here?
```

```
[45]: 'michal'
```

1.8 Exercise (15 min)

Conduct a survey (of your design) using the `input()` function. Produce a formatted report of your user's answers.

The survey should ask questions with floating-point answers and text answers.

HINT: `float(input("Question? "))`

For formatting, have a go at using `print(f"text... {variable}")`

NB. Exercise is best answered along with the exercise from decision-making