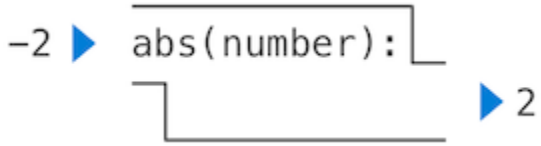
Types of Functions in Python

Throughout this text, we will distinguish between two types of functions.

Pure functions. Functions have some input (their arguments) and return some output (the result of applying them). The built-in function

```
>>> abs(-2)
2
```

can be depicted as a small machine that takes input and produces output.

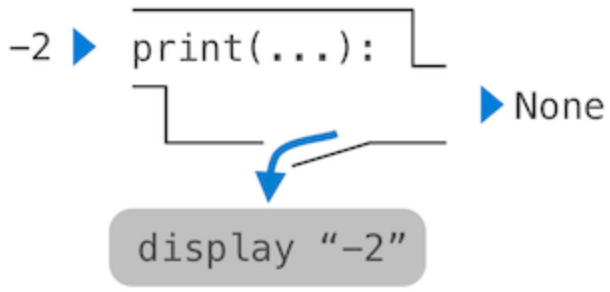


The function `abs` is *pure*. Pure functions have the property that applying them has no effects beyond returning a value. Moreover, a pure function must always return the same value when called twice with the same arguments.

Non-pure functions. In addition to returning a value, applying a non-pure function can generate *side effects*, which make some change to the state of the interpreter or computer. A common side effect is to generate additional output beyond the return value, using the `print` function.

```
>>> print(1, 2, 3)
1 2 3
```

While `print` and `abs` may appear to be similar in these examples, they work in fundamentally different ways. The value that`print` returns is always `None`, a special Python value that represents nothing. The interactive Python interpreter does not automatically print the value `None`. In the case of `print`, the function itself is printing output as a side effect of being called.



A nested expression of calls to `print` highlights the non-pure character of the function.

```
>>> print(print(1), print(2))
1
2
None None
```

If you find this output to be unexpected, draw an expression tree to clarify why evaluating this expression produces this peculiar output.

Be careful with `print`! The fact that it returns `None` means that it *should not* be the expression in an assignment statement.

```
>>> two = print(2)
2
>>> print(two)
None
```

Pure functions are restricted in that they cannot have side effects or change behavior over time. Imposing these restrictions yields substantial benefits. First, pure functions can be composed more reliably into compound call expressions. We can see in the non-pure function example above that `print` does not return a useful result when used in an operand expression. On the other hand, we have seen that functions such as `max`, `pow` and `sqrt` can be used effectively in nested expressions.

Second, pure functions tend to be simpler to test. A list of arguments will always lead to the same return value, which can be compared to the expected return value. Testing is discussed in more detail later in this chapter.

Source: http://inst.eecs.berkeley.edu/~cs61A/book/chapters/functions.html#types-of-functions