03.5_libraries

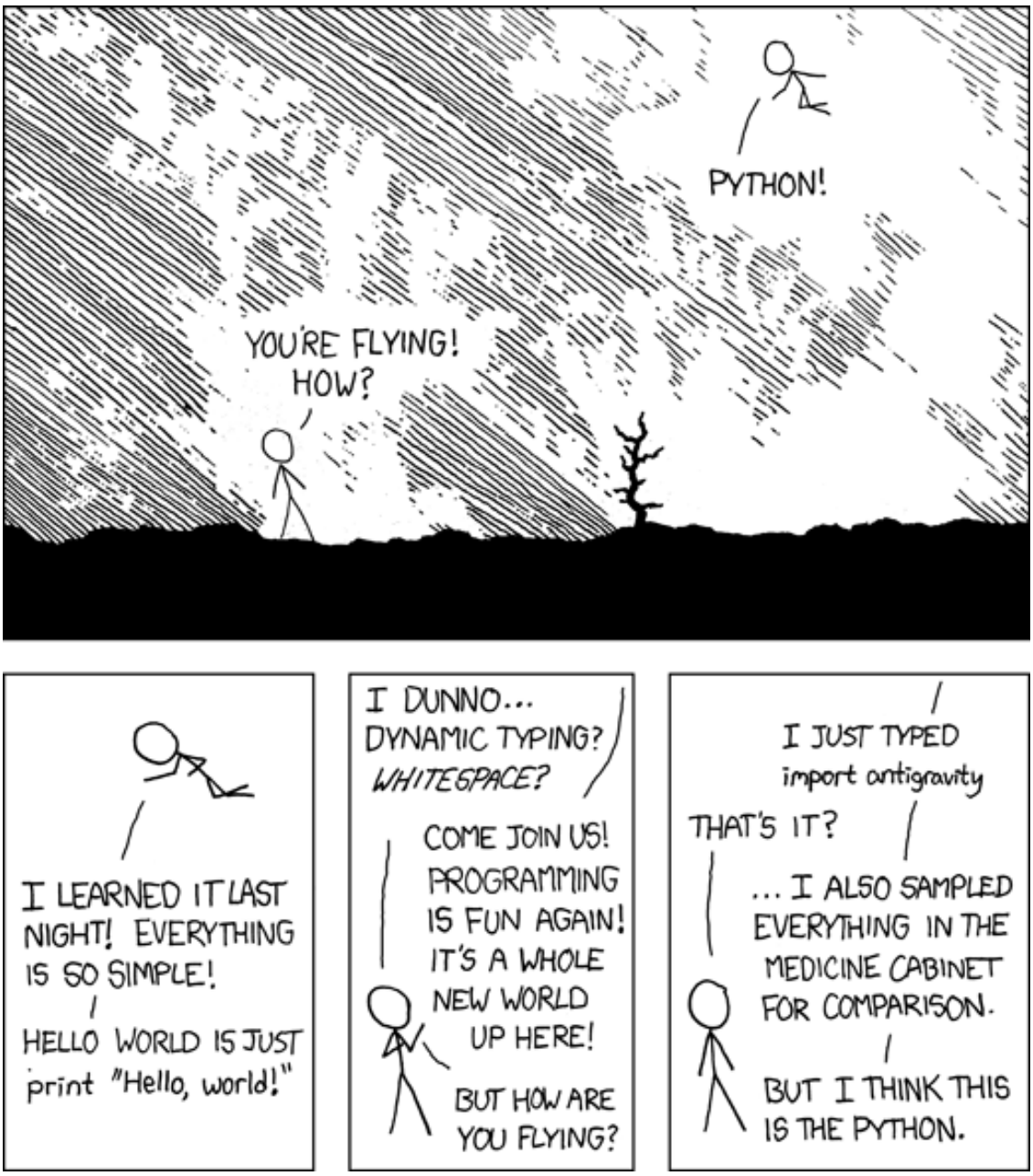
August 12, 2021

1 Libraries

1.1 Most of the power of a programming language is in its libraries.

- A *library* is a collection of files (called *modules*) that contains functions and data values (e.g., numerical constants) for use by other programs.
- The Python standard library is an extensive suite of modules that comes with Python itself.
- Many additional libraries are available from PyPI (the Python Package Index).
- We all have installed a program ('package manager') called `conda` that will also let us get new libraries. We will see this in action later in the course

XKCD Python:



1.2 A program must import a library module before using it.

Use `import` to load a library module into a program's memory.

Then refer to things from the module as `module_name.thing_name` because Python uses `.` to mean "part of".

Using `math`, one of the modules in the standard library:

[1]:
```
import math
```

[2]:
```
print('pi is', math.pi)
```

```
pi is 3.141592653589793
```

[3]:
```
print('cos(pi) is', math.cos(math.pi))
```

```
cos(pi) is -1.0
```

We have to refer to each item with a reference to the module's name.

For example, `math.cos(pi)` won't work because the reference to `pi` doesn't somehow "inherit" the function's reference to `math`.

1.3 Use `help` to learn about the contents of a library module.

This works just like help for a function!

[4]:
```
help(math)
```

```
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.8/library/math

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.
```

```
    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.

    comb(n, k, /)
        Number of ways to choose k items from n items without repetition and
without order.

        Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
        to zero when k > n.

        Also called the binomial coefficient because it is equivalent
        to the coefficient of k-th term in polynomial expansion of the
        expression (1 + x)**n.

        Raises TypeError if either of the arguments are not integers.
        Raises ValueError if either of the arguments are negative.

    copysign(x, y, /)
        Return a float with the magnitude (absolute value) of x but the sign of
y.

        On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(x, /)
        Return the cosine of x (measured in radians).

    cosh(x, /)
        Return the hyperbolic cosine of x.

    degrees(x, /)
        Convert angle x from radians to degrees.

    dist(p, q, /)
        Return the Euclidean distance between two points p and q.

        The points should be specified as sequences (or iterables) of
```

```
        coordinates.  Both inputs must have the same dimension.

        Roughly equivalent to:
            sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))

    erf(x, /)
        Error function at x.

    erfc(x, /)
        Complementary error function at x.

    exp(x, /)
        Return e raised to the power of x.

    expm1(x, /)
        Return exp(x)-1.

        This function avoids the loss of precision involved in the direct
evaluation of exp(x)-1 for small x.

    fabs(x, /)
        Return the absolute value of the float x.

    factorial(x, /)
        Find x!.

        Raise a ValueError if x is negative or non-integral.

    floor(x, /)
        Return the floor of x as an Integral.

        This is the largest integer <= x.

    fmod(x, y, /)
        Return fmod(x, y), according to platform C.

        x % y may differ.

    frexp(x, /)
        Return the mantissa and exponent of x, as pair (m, e).

        m is a float and e is an int, such that x = m * 2.**e.
        If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

    fsum(seq, /)
        Return an accurate floating point sum of values in the iterable seq.

        Assumes IEEE-754 floating point arithmetic.
```

```
    gamma(x, /)
        Gamma function at x.

    gcd(x, y, /)
        greatest common divisor of x and y

    hypot(…)
        hypot(*coordinates) -> value

        Multidimensional Euclidean distance from the origin to a point.

        Roughly equivalent to:
            sqrt(sum(x**2 for x in coordinates))

        For a two dimensional point (x, y), gives the hypotenuse
        using the Pythagorean theorem:  sqrt(x*x + y*y).

        For example, the hypotenuse of a 3/4/5 right triangle is:

            >>> hypot(3.0, 4.0)
            5.0

    isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
        Determine whether two floating point numbers are close in value.

          rel_tol
            maximum difference for being considered "close", relative to the
            magnitude of the input values
          abs_tol
            maximum difference for being considered "close", regardless of the
            magnitude of the input values

        Return True if a is close in value to b, and False otherwise.

        For the values to be considered close, the difference between them
        must be smaller than at least one of the tolerances.

        -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
        is, NaN is not close to anything, even itself.  inf and -inf are
        only close to themselves.

    isfinite(x, /)
        Return True if x is neither an infinity nor a NaN, and False otherwise.

    isinf(x, /)
        Return True if x is a positive or negative infinity, and False
otherwise.
```

```
    isnan(x, /)
        Return True if x is a NaN (not a number), and False otherwise.

    isqrt(n, /)
        Return the integer part of the square root of the input.

    ldexp(x, i, /)
        Return x * (2**i).

        This is essentially the inverse of frexp().

    lgamma(x, /)
        Natural logarithm of absolute value of Gamma function at x.

    log(…)
        log(x, [base=math.e])
        Return the logarithm of x to the given base.

        If the base not specified, returns the natural logarithm (base e) of x.

    log10(x, /)
        Return the base 10 logarithm of x.

    log1p(x, /)
        Return the natural logarithm of 1+x (base e).

        The result is computed in a way which is accurate for x near zero.

    log2(x, /)
        Return the base 2 logarithm of x.

    modf(x, /)
        Return the fractional and integer parts of x.

        Both results carry the sign of x and are floats.

    perm(n, k=None, /)
        Number of ways to choose k items from n items without repetition and
with order.

        Evaluates to n! / (n - k)! when k <= n and evaluates
        to zero when k > n.

        If k is not specified or is None, then k defaults to n
        and the function returns n!.

        Raises TypeError if either of the arguments are not integers.
```

```
        Raises ValueError if either of the arguments are negative.

    pow(x, y, /)
        Return x**y (x to the power of y).

    prod(iterable, /, *, start=1)
        Calculate the product of all the elements in the input iterable.

        The default start value for the product is 1.

        When the iterable is empty, return the start value.  This function is
        intended specifically for use with numeric values and may reject
        non-numeric types.

    radians(x, /)
        Convert angle x from degrees to radians.

    remainder(x, y, /)
        Difference between x and the closest integer multiple of y.

        Return x - n*y where n*y is the closest integer multiple of y.
        In the case where x is exactly halfway between two multiples of
        y, the nearest even value of n is used. The result is always exact.

    sin(x, /)
        Return the sine of x (measured in radians).

    sinh(x, /)
        Return the hyperbolic sine of x.

    sqrt(x, /)
        Return the square root of x.

    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.

    trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.

        Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
```

```
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    /Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/lib-
dynload/math.cpython-38-darwin.so
```

2 In jupyter lab you can also use `shift + tab` to access documentation.

[5]:
```
math.cos  # hit shift tab after the cos
```

[5]:
```
<function math.cos(x, /)>
```

2.1 Import specific items from a library module to shorten programs.

1) Use `from ... import ...` to load only specific items from a library module.
2) Refer to them directly without library name as prefix.

[6]:
```
from math import cos, pi
```

[7]:
```
print('cos(pi) is', cos(pi))
```

```
cos(pi) is -1.0
```

2.2 Create an alias for a library module when importing it to shorten programs.

1) Use `import ... as ...` to give a library a short *alias* while importing it.
2) Refer to items in the library using that shortened name.

[8]:
```
import math as m
```

[9]:
```
print('cos(pi) is', m.cos(m.pi))
```

```
cos(pi) is -1.0
```

You will often see this shortcut with libraries such as NumPy, Pandas, and Matplotlib to shorten your typing. Be careful when you work with others, though, and make sure that you agree on naming conventions in your code.

3 Exercise 03.5

3.1 Practice Importing with Aliases

Example 1 First, let's fill in the following code such that it prints the value of pi / 2 in degrees.

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

[10]:
```
import math as m
angle = m.degrees(m.pi / 2)
print(angle)
```

```
90.0
```

Next, let's rewrite this code such that it uses `import` *without* `as` (i.e., with no alias).

[11]:
```
import math
angle = math.degrees(math.pi / 2)
print(angle)
```

```
90.0
```

Next, let's rewrite this same code using the `from` … `import` … syntax.

[12]:
```
from math import degrees, pi
angle = degrees(pi / 2)
print(angle)
```

```
90.0
```

Which of these three ways is the easiest for you to read? Why would you choose one form over another?