PDSP 2025, Lecture 03, 14 August 2025

Generating sequences of numbers

- `range(n)` generates the sequence `0, 1, 2, ..., n-1`
- Use `list(range(n))` to display as a list

In [1]:
```
n = 17
```

In [2]:
```
range(n)  # Like a list, but not quite
```

Out[2]:
```
range(0, 17)
```

In [3]:
```
list(range(n))  # Make it into a list
```

Out[3]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

- `range(n)` translates to `range(0,n)`, implicitly starting with `0`
- Can add an explicit starting point: `range(i,n)` generates `i,i+1,...,n-1`

In [4]:
```
list(range(2,n))
```

Out[4]:
```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

- If the starting point is $\geq$ the target, `range` generates an empty sequence

In [5]:
```
list(range(3,3))
```

Out[5]:
```
[]
```

In [6]:
```
list(range(7,4))
```

Out[6]:
```
[]
```

Numbers in Python

- Numbers in Python can be integers (`int`) or reals -- actually rationals -- (`float`)
- Internal representation is different, but arithmetic operation symbols are *overloaded* to apply to both types of numbers
- `+`, `-`, `*` stand for addition, subtraction, multiplication, as usual
- `/` is division, and always produces a `float`

In [7]:
```
8/4
```

Out[7]:
```
2.0
```

- There are separate operators for *quotient* (`//`) and *remainder* (`%`)
  - These can also be applied to `float` arguments, but the answer is also `float`

In [8]:
```
8//4
```

Out[8]: 2

In [9]:
```
7 % 3
```

Out[9]: 1

In [10]:
```
8.0//3.0, 8.0 % 5.0
```

Out[10]: (2.0, 3.0)

In [11]:
```
9.3//3.05, 9.3 % 3.05
```

Out[11]: (3.0, 0.15000000000000124)

Data types

- A data type is a set of values with associated operations
- Python has two numeric data types, `int` and `float`
- In the IPL example, we saw text data, which is of type `String` -- we shall examine this later
- The boolean data type has two values `True` and `False`

Checking if a number is prime

- Checking if `n` is a prime: assume it is, and flag that is not if we find a factor between `2` and `n-1`

In [12]:
```
n = 17
isprime = True
for i in range(2,n):
    if n % i == 0:
        isprime = False
```

In [13]:
```
n, isprime
```

Out[13]: (17, True)

In [14]:
```
n = 18
isprime = True
for i in range(2,n):
    if n % i == 0:
        isprime = False
```

In [15]:
```
n, isprime
```

Out[15]: (18, False)

Optimising the search for factors

- Factors occur in pairs, sufficient to check from 2 to $\sqrt{n}$
- Python has a function `sqrt` to compute square roots
- However it is not automatically available

In [16]:
```
sqrt(n)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[16], line 1
----> 1 sqrt(n)

NameError: name 'sqrt' is not defined
```

Libraries

- Libraries are collections of code implementing different groups of functions relevant to a given theme
- We will later see libraries specific to data science, machine learning
- The `math` library has mathematical functions like `sqrt`, `log`, `sin`, `cos` etc
- We `import` the `math` library to use it
  - Note that we use `math.sqrt` to tell Python the full context of the function `sqrt`
  - This is useful in case two different libraries have different functions with the same name

In [17]:
```
import math
```

In [18]:
```
n = 17
math.sqrt(n)
```

Out[18]: 4.123105625617661

Optimised primality checking

- We can optimize our search for factors by restricting the range to `(2,math.sqrt(n))`
- `range` expects only `int` arguments, so use `int()` to convert `math.sqrt(n)` to an `int` -- truncates the fractional part

In [19]:
```
n = 17
isprime = True
for i in range(2,math.sqrt(n)):
    if n % i == 0:
        isprime = False
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[19], line 3
      1 n = 17
      2 isprime = True
----> 3 for i in range(2,math.sqrt(n)):
      4     if n % i == 0:
      5         isprime = False

TypeError: 'float' object cannot be interpreted as an integer
```

In [20]:
```
n = 17
isprime = True
for i in range(2,int(math.sqrt(n))):  # int(...) truncates a float to an
    if n % i == 0:
        isprime = False
```

In [21]:
```
isprime
```

Out[21]:
```
True
```

- We have to be careful, because `range(j,m)` stops at `m-1`
- The code above wrongly claims `25` is a prime -- the search for factors runs from `2` to `4` rather than `2` to `5`

In [22]:
```
n = 25
isprime = True
for i in range(2,int(math.sqrt(n))):  # int(...) truncates a float to an
    if n % i == 0:
        isprime = False
```

In [23]:
```
isprime
```

Out[23]:
```
True
```

- To fix this, modify the upper bound of `range` to `sqrt(n)+1`

In [24]:
```
n = 25
isprime = True
for i in range(2,int(math.sqrt(n))+1):  # int(...) truncates a float to a
    if n % i == 0:
        isprime = False
```

In [25]:
```
isprime
```

Out[25]:
```
False
```

Large and small numbers

- Python allows us to work with very large (and very small numbers)
- The operatoer `**` is exponentiation

In [26]:
```
7**3, 2**12
```

Out[26]:
```
(343, 4096)
```

- What is $2^{2^{12}}$, in other words, 2^{4096}?

In [27]:
```
2**(2**12)
```

Out[27]:
```
1044388881413152506691752710716624382579964249047383780384233483283953907971557456848826811934997558340890106714439262837987573438185793607263236087851365277945956976543709998340361590134383718314428070011855946226376318839397712745672334684344586617496807908705803704071284048740118609114467977783598029006686938976881787785946905630190260940599579453432823469303026696443059025015972399867714215541693835559885291486318237914434496734087811872639496475100189041349008417061675093668333850551032972088269550769983616369411933015213796825837188091833656751221318492846368125550225998300412344784862595674492194617023806505913245610825731835380087608622102834270197698202313169017678006675195485079921636419370285375124784014907159135459982790513399611551794271106831134090584272884279791554849782954323534517065223269061394905987693002122963395687782878948440616007412945674919823050571642377154816321380631045902916136926708342856440730447899971901781465763473223850267253059899795996090799469201774624817718449867455659250178329070473119433165550807568221846571746373296884912819520317457002440926616910874148385078411929804522981857338977648103126085903001302413467189726673216491511131602920781738033436090243804708340403154190336
```

In [28]:
```
2**24
```

Out[28]:
```
16777216
```

In [29]:
```
2**(2**24)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
File ~/python-venv/lib/python3.13/site-packages/IPython/core/formatters.py:770, in PlainTextFormatter.__call__(self, obj)
    763 stream = StringIO()
    764 printer = pretty.RepresentationPrinter(stream, self.verbose,
    765     self.max_width, self.newline,
    766     max_seq_length=self.max_seq_length,
    767     singleton_pprinters=self.singleton_printers,
    768     type_pprinters=self.type_printers,
    769     deferred_pprinters=self.deferred_printers)
--> 770 printer.pretty(obj)
    771 printer.flush()
    772 return stream.getvalue()

File ~/python-venv/lib/python3.13/site-packages/IPython/lib/pretty.py:386, in RepresentationPrinter.pretty(self, obj)
    383 for cls in _get_mro(obj_class):
    384     if cls in self.type_pprinters:
    385         # printer registered in self.type_pprinters
--> 386         return self.type_pprinters[cls](obj, self, cycle)
    387     else:
    388         # deferred printer
    389         printer = self._in_deferred_types(cls)

File ~/python-venv/lib/python3.13/site-packages/IPython/lib/pretty.py:786, in _repr_pprint(obj, p, cycle)
    784 """A pprint that just redirects to the normal repr function."""
    785 # Find newlines and replace them with p.break_()
--> 786 output = repr(obj)
    787 lines = output.splitlines()
    788 with p.group():

ValueError: Exceeds the limit (4300 digits) for integer string conversion; use sys.set_int_max_str_digits() to increase the limit
```

- How about 2^{-4096}?

In [30]:
```
2**(-(2**12))
```

Out[30]:
```
0.0
```

- The value has become too small to distinguish from zero
- On the other hand 2^{-1024} works

In [31]:
```
2**(-(2**10))
```

Out[31]:
```
5.562684646268003e-309
```