







Item	Python Pint	raku Physics::Measure	Comments
Load package/ module	import pint ureg = pint.UnitRegistry()	use Physics::Measure :ALL;	
Quick start	3 * ureg.meter + 4 * ureg.cm <Quantity(3.04, 'meter')>	3m + 4cm; 3.04m	Concise Raku postfix operators are defined for all SI prefix + unit combos
Define Quantity / Measurement	book_length = (20. * ureg.centimeter). plus_minus(2.) print(book_length) (20.0 +/- 2.0) centimeter	my \book-length = 20cm ±2; say book-length; 20cm ±2	Pint and Raku both handle measurement uncertainty / error
Absolute & relative errors	book_length = (20. * ureg.centimeter). plus_minus(.1, relative=True) print(book_length) (20.0 +/- 2.0) centimeter	say book-length = 20cm ±10%; 20cm ±2	Raku also does percent error
Math operations with Errors	print(2 * book_length) (40 +/- 4) centimeter	2 * book-length; 40cm ±4	Error is adjusted linearly depending on the operation
Format output	print('{:.02fP}'.format(book_length)) (20.00 ± 2.00) centimeter	n/a	Pint applies formatting to both value and error
Control rounding with Error	n/a	my \$val1 = 2.8275793cm ±5%; 2.8276cm ±0.141 my \$val2 = 2.8275793cm 2.8275793cm	Raku auto-adjusts rounding of value to reflect the accuracy of the Error < here it is without an Error for comparison (

Math operations with Measures	distance = 24.0 * ureg.meter 24.0 meter time = 8.0 * ureg.second 8.0 second speed = distance / time 3.0 meter / second	my \$distance = 24m; 24m my \$time = 8s; 8s my \$speed = \$distance/\$time; 3m/s	Both packages automatically derive the dimensions and result type from the arguments.
Dimension Errors	print(distance + time) DimensionalityError Traceback (most recent call last)...	say \$distance + \$time; cannot convert in to different type Length...	Both pint and raku protect against combination errors.
Variable Types	speed = time / distance print(speed) 0.333333333333333 second / meter	my Speed \$speed = \$time / \$distance; No such symbol	raku Physics::Measure types are fully fledged class types and integrate with the raku type system so they protect against a wider range of errors (ok this error message is LTA)
Type System Integration	n/a	given \$measurement { when Length { say 'long' } when Time { say 'long' } when Speed { say 'fast' } } - or - subset Limited of Speed where 0 <= *.in('mph') <= 70;	Integration with the raku type system means that Physics::Measure types get the full potential of type-oriented language features (this raku switch statement is just one example) Another example is the class subset with a where constraint
Type Conversion	speed.to('inch/minute') <Quantity(7086.61417, 'inch minute')>	\$speed .= in('inch/minute'); 7086.614173inch/minute	Both systems can convert to a very wide range of compound units.
Type Derivation	energy = ureg('68 J') time = ureg('8 s') power = energy/time	my \$energy = 68J; my \$time = 8s; say \$energy / \$time;	raku automatically applies the SI Derived Unit relationships

	print(power.to_compact()) 8.5 joule / second	8.5W	
Parsing Unit Strings	height = ureg('10 ft') 10 foot	my \$height =  '10 ft'; 10ft	Both systems can read natural language units strings. [the  prefix feeds a raku Grammar]
Parsing Gamut	> m meter metre meter > ft foot feet foot > km kilom kilometer kilometer > J joule joules joule > kg m^2 s^2 kg m^2/s^2 kg m**2/s**2 kg·m²/s² kilogram·meter²/second²	> m meter metre m > ft foot feet ft > km kilom kilometer km > J joule joules J > kg m^2 s^2 kg m^2/s^2 kg m**2/s**2 kg·m²/s² J	Both handle a similar input range: <ul style="list-style-type: none"> • SI, US and Imperial • Prefix & Unit name • Abbreviations & Plurals • Power symbols ^ ** ² Raku output defaults to the abbreviated initial(s) Raku knows that "kg m^2/s^2" is the same as "J" and automatically applies the SI Derived Unit relationship
Angles & Trigonometry	n/a	my \θ1 =  <45°30'30">; 45°30'30" my \sine = sin(θ1); 0.7133523847299412	Raku implements a domain specific format for degrees / minutes / seconds and all trig functions (including radians)
Number Precision	thickness = 68 * ureg.m area = 60 * ureg.km**2 n2g = 0.5 * ureg.dimensionless phi = 0.2 sat = 0.7 volume = area * thickness * n2g * phi * sat	my \thickness = 68m; my \area =  '60 sq km'; my \n2g =  '0.5 ①'; my \φ = 0.2; my \sat = 0.7; my \volume = area * thickness * n2g * φ * sat;	Pint has optional units.dimensionless ... raku has optional unicode for this ① and for variable names like phi(φ) (raku has flexible whitespace)

	285.5999999999997 kilometer ² meter	285600000m ³	raku uses Rat number types to avoid this kind of float-induced trailing decimals
Compacting	volume.to_compact() 285599999.99999994 meter ³	n/a	raku Physics::Unit automatically chose the compact form already
Normalising	volume.to_compact('L') 285.6 gigaliter	volume.in('l').norm; 285.6Gl	In raku we use .in to convert the units and then .norm to adjust to the most convenient prefix
Formatting	'The pretty representation is {:~P}'.format(accel) 'The pretty representation is 1.3 m/s ² '	"The pretty representation is {\$accel.pretty}" 'The pretty representation is 1.3 m·s ⁻² '	Raku .pretty applies the formal SI unicode standard. Pint has Latex and HTML output options.
Temperature conversion	ureg.default_format = '.3f' Q_ = ureg.Quantity home = Q_(25.4, ureg.degC) print(home.to('degF')) 77.720 degree_Fahrenheit	my \degC =  '25.4 °C'; my \degF = degC.in('°F'); say degF; 77.72°F	Raku use of unicode '°' degree symbol makes temperatures very natural.
Comparison	n/a	my \$a = 4.3m; my \$af = \$a.in: 'feet'; 14.108 feet say \$af cmp \$a; Same	Raku numeric and string comparisons work, with automatic Unit conversion and allowance for ± Errors

<https://pint.readthedocs.io/en/stable/measurement.html>

<https://pint.readthedocs.io/en/stable/tutorial.html>