**Python** –

*Hours to Minutes:*

minutes = int(input('Enter minutes:\n'))

hours = minutes // 60

minutes\_remaining = minutes % 60

print(minutes, 'minutes is', end=' ')

print(hours, 'hours and', end=' ')

print(minutes\_remaining, 'minutes.\n', end=' ')

Given a 10-digit phone number stored as an integer, % and / can be used to get any part, such as the prefix. For phone\_num = 9365551212 (whose prefix is 555):

tmp\_val = phone\_num // 10000 # // 10000 shifts right by 4, so 936555.

prefix\_num = tmp\_val % 1000 # % 1000 gets the right 3 digits, so 555.

Dividing by a power of 10 shifts a value right. 321 // 10 is 32. 321 // 100 is 3.

% by a power of 10 gets the rightmost digits. 321 % 10 is 1. 321 % 100 is 21.

ones\_digit = user\_val % 10 # Ex: 927 % 10 is 7.

tmp\_val = user\_val // 10

tens\_digit = tmp\_val % 10 # Ex: tmp\_val = 927 // 10 is 92. Then 92 % 10 is 2.

tmp\_val = tmp\_val // 10

hundreds\_digit = tmp\_val % 10 # Ex: tmp\_val = 92 // 10 = 9. Then 9 % 10 is 9.

Checking if a file was executed as a script.

|  |
| --- |
| # The baby\_names.py module  **print** ('Initializing baby variables...')  baby\_name1 = 'Ryder'  baby\_name2 = 'Jess'  baby\_weight1 = 5.1  baby\_weight2 = 8.5  # Executes only if file run as a script (e.g., python baby\_names.py)  **if** \_\_name\_\_ == '\_\_main\_\_':  **print**('Baby 1:', baby\_name1, 'was born', baby\_weight1, 'lbs')  **print**('Baby 2:', baby\_name2, 'was born', baby\_weight2, 'lbs')    # A script favorite\_child.py that imports and uses the baby\_names module.  **import** baby\_names # Importing the module executes the module contents  **print**('My favorite child is', baby\_names.baby\_name1, '-')  **print**('I remember when he weighed only', baby\_names.baby\_weight1, 'pounds.')  **print**('I love', baby\_names.baby\_name2, 'too, of course.') |

**Savings Interest Program**

import math

base = float(input('Enter initial savings: '))

print()

rate = float(input('Enter annual interest % rate: '))

print()

years = int(input('Enter years that pass: '))

print()

total = base \* math.pow(1 + (rate / 100), years)

print ('Savings after', years, 'years is', total)

Functions in the standard math module.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | Description |  | Function | Description |
| ceil | Round up value |  | fabs | Absolute value |
| factorial | factorial (3! = 3 \* 2 \* 1) |  | floor | Round down value |
| fmod | Remainder of division |  | fsum | Floating-point sum |
| exp | Exponential function ex |  | log | Natural logarithm |
| pow | Raise to power |  | sqrt | Square root |
| acos | Arc cosine |  | asin | Arc sine |
| atan | Arc tangent |  | atan2 | Arc tangent with two parameters |
| cos | Cosine |  | sin | Sine |
| hypot | Length of vector from origin |  | degrees | Convert from radians to degrees |
| radians | Convert degrees to radians |  | tan | Tangent |
| cosh | Hyperbolic cosine |  | sinh | Hyperbolic sine |
| gamma | Gamma function |  | erf | Error function |
| pi (constant) | Mathematical constant 3.141592... |  | e (constant) | Mathematical constant 2.718281... |

Escape SequenceExplanationExample codeOutput\\Backslash (\)

**print**('\\home\\users\\')

\home\users\

\'Single quote (')

**print**('Name: John O\'Donald')

Name: John O'Donald

\"Double quote (")

**print**("He said, \"Hello friend!\".")

He said, "Hello friend!".

\nNewline

**print**('My name...\nIs John...')

My name...

Is John...

\tTab (indent)

**print**('1. Bake cookies\n\t1.1. Preheat oven')

1. Bake cookies

1.1. Preheat oven

#A 'Mad Libs' style game where user enters nouns,

#verbs, etc., and then a story using those words is ouput.

#Get user's words

relative = input('Enter a type of relative: ')

print()

food = input('Enter a type of food: ')

print()

adjective = input('Enter an adjective: ')

print()

period = input('Enter a time period: ')

print()

# Tell the story

print('My', relative, 'says eating', food)

print('will make me more', adjective)

print('so now I eat it every', period)

alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

user\_number = int(input('Enter number to use as index: '))

print()

print('\nLetter', user\_number, 'of the alphabet is', alphabet[user\_number])

# Some of the most expensive cars in the world

lamborghini\_veneno = 3900000 # $3.9 million!

bugatti\_veyron = 2400000 # $2.4 million!

aston\_martin\_one77 = 1850000 # $1.85 million!

prices = [lamborghini\_veneno, bugatti\_veyron, aston\_martin\_one77]

**print**('Lamborghini Veneno:', prices[0], 'dollars')

**print**('Bugatti Veyron Super Sport:', prices[1], 'dollars')

**print**('Aston Martin One-77:', prices[2], 'dollars')

|  |  |
| --- | --- |
| Operation | Description |
| len(list) | Find the length of the list. |
| list1 + list2 | Produce a new list by concatenating list2 to the end of list1. |
| min(list) | Find the element in list with the smallest value. |
| max(list) | Find the element in list with the largest value. |
| sum(list) | Find the sum of all elements of a list (numbers only). |
| list.index(val) | Find the index of the first element in list whose value matches val. |
| list.count(val) | Count the number of occurrences of the value val in list. |

GRADE CALCULATOR

exam1\_grade = float(input('Enter score on Exam 1 (out of 100):\n'))

exam2\_grade = float(input('Enter score on Exam 2 (out of 100):\n'))

exam3\_grade = float(input('Enter score on Exam 3 (out of 100):\n'))

overall\_grade = (exam1\_grade + exam2\_grade + exam3\_grade) / 3

print('Your overall grade is:', overall\_grade)

|  |  |  |
| --- | --- | --- |
| Replacement definition | Example | Formatted string result |
| Positional replacement | 'The {1} in the {0} is {2}.'.format('hat', 'cat', 'fat') | The cat in the hat is fat. |
| Inferred positional replacement | 'The {} in the {} is {}.'.format('cat', 'hat', 'fat') | The cat in the hat is fat. |
| Named replacement | 'The {animal} in the {headwear} is {shape}.'.format(animal='cat', headwear='hat', shape='fat') | The cat in the hat is fat. |

A ***format specification*** inside of a replacement field allows a value's formatting in the string to be customized. Ex: Using a format specification, a variable with the integer value 4 can be output as a floating-point number (4.0) or with leading zeros (004).

A common format specification is to provide a ***presentation type*** for the value, such as integer (4), floating point (4.0), fixed precision decimal (4.000), percentage (4%), binary (100), etc. A presentation type can be set in a replacement field by inserting a colon : and providing one of the presentation type characters described below.

Table 2.7.2: Common formatting specification presentation types.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Description | Example | Output |
| s | String (default presentation type - can be omitted) | '{:s}'.format('Aiden') | Aiden |
| d | Decimal (integer values only) | '{:d}'.format(4) | 4 |
| b | Binary (integer values only) | '{:b}'.format(4) | 100 |
| x, X | Hexadecimal in lowercase (x) and uppercase (X) (integer values only) | '{:x}'.format(15) | f |
| e | Exponent notation | '{:e}'.format(44) | 4.400000e+01 |
| f | Fixed-point notation (6 places of precision) | '{:f}'.format(4) | 4.000000 |
| .[precision]f | Fixed-point notation (programmer-defined precision) | '{:.2f}'.format(4) | 4.00 |
|  |  |  |  |

Replacement typeExampleOutputInferred positional replacement

'{:s} ${:.2f} tacos is ${:.2f} total'.format('Three', 1.50, 4.50)

Three $1.50 tacos is $4.50 total

Positional replacement

'{0:s} ${2:.2f} tacos is ${1:.2f} total'.format('Three', 4.50, 1.50)

Three $1.50 tacos is $4.50 total

Named replacement

'{cnt:s} ${cost:.2f} tacos is ${sum:.2f} total'.format(cnt = 'Three', cost = 1.50, sum = 4.50)

Three $1.50 tacos is $4.50 total

A format specification can include an ***alignment character*** that determines how a value should be aligned within the width of the field. Alignment is set in a format specification by adding a special character before the field width integer. The basic set of possible alignment options include left-aligned '<', right-aligned '>' and centered '^'.

Figure 2.8.2: Aligning strings within a field.

Consider the following code that prints a table, and how changing the alignment impacts the column organization.

format\_string = <format\_string> # Replaced in table below

**print**(format\_string.format(name='Player Name', goals='Goals'))

**print**('-' \* 24)

**print**(format\_string.format(name='Sadio Mane', goals=22))

**print**(format\_string.format(name='Gabriel Jesus', goals=7))

|  |  |  |
| --- | --- | --- |
| Alignment type | <format\_string> | Output |
| Left-aligned | '{name:<16}{goals:<8}' | Player Name Goals  ------------------------  Sadio Mane 22  Gabriel Jesus 7 |
| Right-aligned | '{name:>16}{goals:>8}' | Player Name Goals  ------------------------  Sadio Mane 22  Gabriel Jesus 7 |
| Centered | '{name:^16}{goals:^8}' | Player Name Goals  ------------------------  Sadio Mane 22  Gabriel Jesus 7 |

A list of common slicing operations a programmer might use.  
Assume the value of my\_str is 'http://en.wikipedia.org/wiki/Nasa/'

|  |  |  |
| --- | --- | --- |
| Syntax | Result | Description |
| my\_str[10:19] | wikipedia | Gets the characters in indices 10-18. |
| my\_str[10:-5] | wikipedia.org/wiki/ | Gets the characters in indices 10-28. |
| my\_str[8:] | n.wikipedia.org/wiki/Nasa/ | All characters from index 8 until the end of the string. |
| my\_str[:23] | http://en.wikipedia.org | Every character up to index 23, but not including my\_str[23]. |
| my\_str[:-1] | http://en.wikipedia.org/wiki/Nasa | All but the last character. |

|  |  |  |
| --- | --- | --- |
| Example | Expression result | Why? |
| 'Hello' == 'Hello' | True | The strings are exactly identical values |
| 'Hello' == 'Hello!' | False | The left hand string does not end with '!' |
| 'Yankee Sierra' > 'Amy Wise' | True | The first character of the left side 'Y' is "greater than" (in ASCII value) the first character of the right side 'A' |
| 'Yankee Sierra' > 'Yankee Zulu' | False | The characters of both sides match until the second word. The first character of the second word on the left 'S' is not "greater than" (in ASCII value) the first character on the right side 'Z' |
| 'seph' in 'Joseph' | True | The substring 'seph' can be found starting at the 3rd position of 'Joseph' |
| 'jo' in 'Joseph' | False | 'jo' (with a lowercase 'j') is not in 'Joseph' (with an uppercase 'J') |