1 Basics of Python Programming

1.1 Types of errors
- Syntax errors
- Runtime errors
- Logic errors

1.2 Python Objects
- Everything created in Python is implemented as an **Object**
- Objects are pieces of memory, with values and associated operations

1.2.1 Types of objects
- Can find out data type of Python objects using the `type()` command
- e.g. `int`, `str`, `float`, `bool`
- Types of object are important as they define the possible values of objects as well as the operations that the object supports. e.g. 2.3 + 5 ⟶ 7.3, `'Hello' + 'World'` ⟶ `'Hello World'`, `'Hello' * 2` ⟶ `'Hello Hello'`
- Note that you cannot mix data types in some operations e.g. `'Hello' + 5` will result in `type error`

1.2.2 Data type conversion
- Enables other operations by another data type
- Use the type name as the data type conversion

```python
num = '5'
print(type(num)) # <class 'str'>
num_int = int(num)
print(type(num_int)) # <class 'int'>
num_float = float(num)
print(type(num_float)) # <class 'float'>
```

- Typically used when dealing with user input and we want to convert from string to other data types for manipulation
- Note that data conversion does not actually change the type of the original object! It **creates a new object with the same value of a different type**

1.3 Variables and Assignment Operation
- Syntax for variable names:
  - Only 1 word
  - Only consist of letters, numbers, and underscores
  - Cannot begin with a number
  - Avoid contradictions with Python keywords

2 Control Flows

2.1 if-else Ternary Expression
- e.g. sold = demand if demand < order else order is a substituted for

3 Built-in Data Structures

3.1 Strings

3.1.1 How to create strings
- Can create a new string object either by using single or double quotes e.g. `'Hello'` or `"Hello"`
- Multi-line strings can be created using 3 single/double quote, all indentation will be preserved e.g.

```python
shining = """
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
    All work and no play
    makes Jack a dull boy
    All work and no play
    makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
"""
```

- output of `input()` function
- data type conversion using `str()` function
- concatenate or duplicate other strings

3.1.2 String Methods
- Length of string can be found using the `len()` function e.g. `len("Hello")` = 5
- Case conversion methods
  - `string.upper()` - changes all character to upper case
  - `string.lower()` - changes all characters to lower case
  - `string.capitalize()` - capitalizes the first word of the string
  - `string.swapcase()` - swaps between upper and lower case of each character in the original string
  - `string.title()` - capitalizes first character of each word in string
- `count(substr)` - returns the number of occurrence of a substring
- `replace(original, new)` - returns a copy with all occurrence of original substr replaced by new substr

3.1.3 String Indexing & Slicing
- Strings are 0-indexed
- Can alternatively be negatively indexed ⟶ last character starts with index of −1
- Slicing of string have the syntax `[start:stop:step]` with last character being the character before stop
- Possible to use slicing to reverse a string by doing `string[::-1]`

3.2 Lists

3.2.1 Creation of lists
- `furious_five = ['Tigress', 'Crane', 'Mantis', 'Monkey', 'Viper']`
- Can also mix data types in lists e.g. `numbers = [1, 2.0, 3.0, 4, 5, 6.0]`
- Possible to have empty lists i.e. []
- Can create list through type conversion e.g. `list('abcd')` = ['a', 'b', 'c', 'd'] or `list(range(5))` = [0, 1, 2, 3, 4]
- List comprehension: `[expression for item in iterable if conditions]` e.g. `[word for word in words if word[0].lower() in 'aeiou']`
- Note that `list1 = list2` does not actually copy list2 into list1!!. It just copies a reference to the list ⟶ any changes made to list2 will affect list1

3.2.2 Similarities between Lists and Strings
- uses the same `len()` function
- uses the same indexing and slicing system

3.2.3 Datatypes of list slicing
- Following the example in the above image, `type(arr[2])` = str, `type(arr[2:3])` = list and `type(arr[2:2])` = list (empty list)

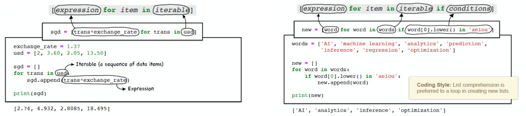
3.2.4 Difference between String and List
- Mutability

```python
my_answers = ['B', 'C', False, True, 0.256, 2]
print(my_answers)

my_answers[1] = 'D'
print(my_answers)

my_answers[2:4] = [True, False]
print(my_answers)
```

```
['B', 'C', False, True, 0.256, 2]
['B', 'D', False, True, 0.256, 2]
['B', 'D', True, False, 0.256, 2]
```

3.2.5 List Methods
- `append(item)` - add items to the back of the list
- `extend(list)` - append items from list to another list
- `insert(pos, item)` - insert item at index pos ⟶ insert new element at pos and shift all elems from pos onwards to the right by 1
- `remove(item)` - search and removes the first appearance of an item in a list, error message is raised if given value is not in list
- `pop(index)` - removes and returns the item at index specified, default to last item in list
- `index(item)` - search and returns the index of first appearance of item in list, error raised if item not in list

3.2.6 List Comprehension



3.3 Tuples

3.3.1 Tuple Creation

```python
colors = 'red', 'blue', 'green'
mixed = ('Jack', 32.5, [1, 2])
feel_empty = () # empty tuple
tuple_one = 'here', # tuple
item_one = ('there') # string
```

3.3.2 Features of tuple
- Immutable
- Iterable
- Same `len()` function, indexing, slicing as strings and lists
- Same + and * as strings and lists
- No method is defined for tuple
- Tuple unpacking

```python
location = 1.290, 103.852
latitude, longitude = location

print(latitude)
print(longitude)
```

```
1.29
103.852
```

```python
x, y, z = 'abc'

print(x)
print(y)
print(z)
```

```
a
b
c
```

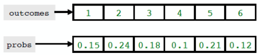
- `zip()` function
  - can be applied to other iterable types
  - can be used to more than 2 data sequences
  - For sequences with different lengths, the iteration stops when the shortest sequence is running out of items

```python
for item in zip(outcomes, probs):
    print(item)
```

```
(1, 0.15)
(2, 0.24)
(3, 0.18)
(4, 0.1)
(5, 0.21)
(6, 0.12)
```

```python
print(type(zip(outcomes, probs)))
```

```
<class 'zip'>
```

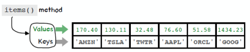


3.4 Dictionaries
- Comma-separated *key* : *value* pairs enclosed in curly brackets
- Keys can be any immutable types: boolean, integer, float, string, tuple, etc
- Values can be any data type
- Mutable

3.4.1 Dictionary methods
- Iterating over key-value items

```python
for item in stocks.items():
    print(item)
```

```
('AMZN', 170.40)
('TSLA', 130.11)
('TWTR', 32.48)
('AAPL', 76.6)
('ORCL', 51.58)
('GOOG', 1434.23)
```



- `items()` will get you a (key, value) tuple
- `keys()` will get you all the keys in a list
- `values()` will get you all the values in a list

3.5 Summary of data structures

	String	List	Tuple	Dictionary
mutable	No	Yes	No	Yes
indexing and slicing	integers	integers	integers	key names
operators + and *	Yes	Yes	Yes	No
iterable	Yes	Yes	Yes	Yes
methods	Yes	Yes	No	Yes

	`()`	`[]`	`{}`
Usage	1. Enclose input arguments of function and method 2. Create tuples	1. Create lists 2. Indexing and slicing	1. Dictionary and Set 2. Used in f-strings or `format()` method

Examples	`print('Hello')`
	`string.upper()`
	Empty tuple `()`
		`[1, 2, 3]`
		`string[3:]`
		`dictionary['key']`
			`{'key': 'value'}`

Remarks	1. Cannot be omitted even when there is no input argument 2. Can be omitted when creating tuples	-	Set is not covered in this course

4 Function, Modules, Packages

4.1 Syntax for function names
- Only 1 word
- Only consist of letters, numbers, and underscores
- Cannot begin with a number
- Avoid contradictions with Python keywords

4.2 Syntax of function definition
- Function terminates when it hits `return` - will not run any code after `return` statement
- If a function doesn't have a return statement, it implicitly returns `null`

4.3 Benefits of functions
1. Reuse code
2. Easy to test and debug
3. Higher readability

4.4 Function arguments and outputs

4.4.1 Positional arguments, keyword arguments and default values

```python
student = student_info('Jane Doe', 20,
                       year=2019, faculty='Business', gender='F')
```

Positional arguments

Keyword arguments

```python
def student_info(name, age, gender, faculty, year):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

Notes: In Python programming, all positional arguments must be specified before keyword arguments, otherwise there will be an error message.

4.4.2 Multiple Outputs

```python
def student_info(name, age, gender, faculty='Business', year=2019):
    """
    This function summarizes student info. into a dictionary and
    return the generated email for the student.
    """

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    lower_name = name.lower().replace(' ', '_')
    lower_faculty = faculty.lower()
    email = '{}_{}@u.nus.edu.sg'.format(lower_name, lower_faculty)

    return info_dict, email
```

A tuple!!

4.5 Scopes and namespaces
- Namespace is a collection of names
- The same name may be used to represent different objects in different scopes
- Scopes can be defined as follows:
  - **Local**: input arguments and names declared (by assignment statements) in a function and are effective only within the function
  - **Global**: names declared outside of function definitions
  - **Built-in**: names predefined in Python

```python
age = [11, 36, 37, 67]

def zoo_ticket(age):
    """The function zoo_ticket calculates the
    Singapore Zoo ticket prices for visitors"""

    if age < 3:
        ticket = 0
    elif age <= 12:
        ticket = 28
    elif age < 60:
        ticket = 41
    else:
        ticket = 18

    return ticket

ticket = [zoo_ticket(item) for item in age]
```

Global name "age" is a list

Global scope

Local name "age" is an integer

Local scope

Global name "age" is a list

Global scope

4.6 Modules
A module is a ".py" file that defines functions, classes, variables, or simply contains some runnable code.

4.6.1 Benefits of Modules
- Code reuse
- System namespace partitioning
- Implementing shared services or data

4.6.2 Syntax of importing modules

from Module name import Object name as Alias

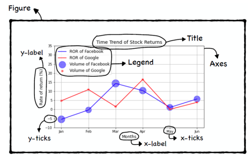
Alias

```python
from statistics import mean as me
from statistics import variance as var

numbers = [0.5, 1.2, 3.8, 2.4, 4.1, 3.5]

print(me(numbers))
print(var(numbers))
```

```
2.5833333333333333
2.1816666666666666
```

4.7 Packages
- A collection of modules and supporting files
- Use `.` operator to indicate the directory hierarchy

Plot feature	Argument keyword	Plot function
Color	`color`	`plot()`, `bar()`, `scatter()`
Opacity	`alpha`	`plot()`, `bar()`, `scatter()`
Marker shape	`marker`	`plot()`, `scatter()`
Line style	`linestyle`	`plot()`
Line width	`linewidth`	`plot()`
Bar width	`width`	`bar()`

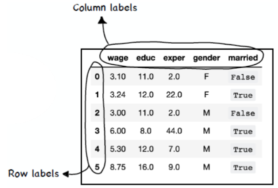


5 Lovely Pandas

5.1 Data Representation
- Variables (fields/attributes) as columns
- Observations (cases/records) as rows

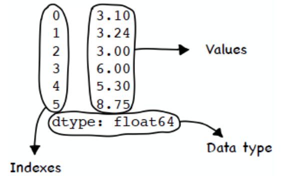
5.1.1 Types of Variable
- **Numerical (Quantitative)** - e.g. things that can be represented with float, int
- **Categorical (qualitative)** - e.g. things that are represented using strings ("M/F"), True/-False
- Note that True/False can be converted into numerical form by doing `int(True/False)` which gives 1/0 respectively

5.1.2 Types of Data Situations
- Could have labeled data
- Heterogeneous data - mix of datatypes in the same datasets
- Possible missing values - represented by NA in dataset

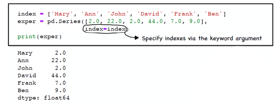


5.2 pandas.Series
- 1-dimensional array of indexed data e.g. 1 row or 1 col of dataset
- Can be created from a list e.g. wage = `pd.Series([3.10, 3.24, 3.00, 6.00, 5.30, 8.75])` or from a tuple e.g. educ = `pd.Series((11.0, 12.0, 11.0, 8.0, 12.0, 16.0))`
  - Note that the data type of the series will be automatically converted to `float64`

5.2.1 Attributes



Index
- accessed using the `series.index` attribute
- Can specify indexes using an array of values
- `series.index` returns for e.g. `Index(['Mary', 'Ann', 'John', 'David', 'Frank', 'Ben'], dtype='object')`

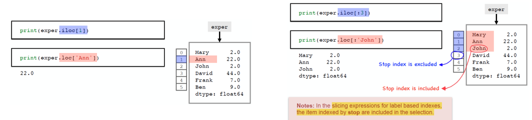


dtypes
- `series.dtype` will return the datatype of the values in the series
- In this course: No difference between 64 bit and 32 bit values

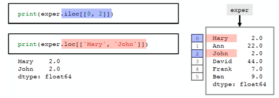
Pandas dtype	Built-in Python types
object	str or mixed types
int64	int
float64	float
bool	bool

- can convert the dtype to other types by using the `series.astype(<type>)` method

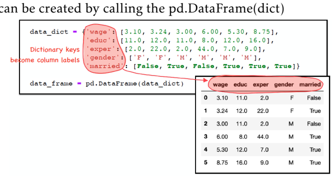
5.2.2 Series indexing and Slicing
- indexing and slicing is done by the `series.iloc[]` and `series.loc[]` methods
- Slicing using `iloc[]` method does not include the selection index by stop index but slicing using the `loc[]` method includes the selection index by end index



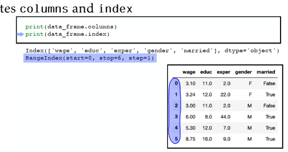
- Can also select specific rows/indexes to extract out



5.3 pandas.DataFrame
- Dataframe can be created by calling the pd.DataFrame(dict)



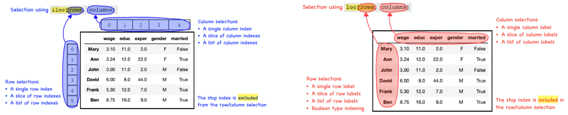
- Has the attributes `columns` and `index`



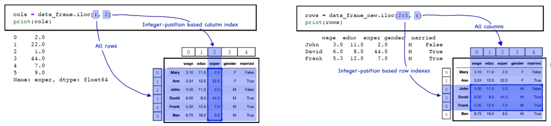
- Can set the index similarly to series
- Has the `dtypes` attribute which returns the `dtype` of each of the columns

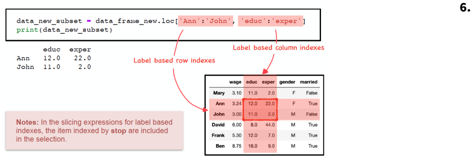
5.3.1 Slicing and Indexing of Dataframe

- Note that the **default value will be all columns** if columns not specified



Slicing all rows and specific column / all columns specific rows



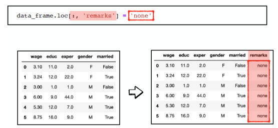


Indexing of Column

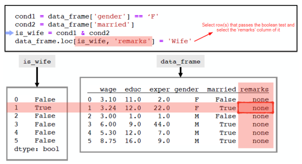
- columns of dataframes can be indexed using dataframe['<col_name>']

5.3.2 Mutating Dataframes in-place

- `data_frame.loc[2:3, 'educ'] = 9.0 # modifies col 'educ' of rows 2 and 3 to the value 9`
- `data_frame.iloc[2, 1:3] = 1.0 # changes cols 1 to 2 of row 2 to value of 1.0`
- Adding a new column



5.3.3 Filtering Data



- Includes other bit-wise manipulation among conditions such as | for OR operation and ~ for NOT operation
- Not operation could alternatively be -ve of the negation e.g. `cond1 = data_frame['gender'] == 'F'` `is_male = -cond1`

5.4 Pandas Methods

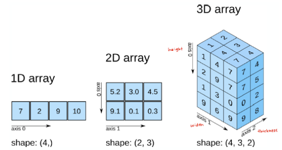
- `pd.read_csv(csv_file)` → reads a cvs file into a pandas dataframe
- `dataframe.head(6)` → shows first 6 records
- `dataframe.drop(columns='gender')` → drop the column labeled 'gender'
- The following only works if columns are all quantitative values
  - `dataframe.mean()/median()/mode()` → returns the mean/median/mode of each column
  - `dataframe.var()/std()` → returns the variance/standard deviation of each column
- `dataframe.min()/max()` → returns the min/max values of each column, if comparing **string** we compare it lexicographically, **boolean** max will be True and min will be False
- `dataframe[<col>].value_counts()` → returns the count (frequencies) of all unique values
- `dataframe[<col>].value_counts(normalize=True)` → returns the proportions of all unique values, values will add up to 1
- `dataframe.corr()/cov()` → returns the correlation/covariance between each column
- `dataframe.describe()` → returns count, mean, std, min, 25%, 50%, 75% and max for each col
- `dataframe[<col>].isnull()/notnull()` → returns whether values in col is null/not-null
- `dataframe.dropna()` → returns a copy of the data frame without any rows of missing values. Original dataframe remains unchanged. If we want to modify original dataframe, use the `inplace=True` argument
- `dataframe.fillna(<val>)` → returns a copy of the dataframe with all NaN values replaced by val, use `inplace=True` to modify in place
- `dataframe.set_index(<col>)` → sets col to be the index of dataframe
- `dataframe.reset_index` → typically used to rest the integer index after `dropna()` or filtering, if `set_index()` called previously, previous label turns into a column. If `reset_index()` called after `dropna()` the previous integer indexes also becomes a column. Use `drop=True` to drop previous labels

5.4.1 Element-wise arithmetic operations

- Allows us to perform arithmetic operations on each of the elements in the dataframe without loops e.g.
  `gdp_new['1980'] / 1000000 # divide each values in col '1980' by 1000000`
- Possible to perform arithmetic operations on 2 cols e.g.
  `gdp_new['1981'] - gdp_new['1980'] # takes every value of col '1981' and subtract from it the corresponding value in col '1980'`
- To perform string operations on values:
  ```
  project = prop['project']
  project_lower = project.str.lower() # changes all values in col
  ↪ 'project' to lowercase
  ```
- Possible to use any string operation after the `.str`
- Indexing using the `.str`: e.g.
  `prop['level_from'] = prop['level'].str[:2].astype(int)`

6 Numpy

- Supports the representation of array-like objects like range, list, tuple, series and dataframe



- pandas uses numpy internally e.g. `type(dataframe.values)` is actually of type `<class 'numpy.ndarray'>`

6.1 Attributes of Numpy

- `ndim` → dimension number of the array
- `shape` → shape of array, given as **tuple**, e.g. 1-dim array will have value (6,), 2-dim array will have value (6, 5). Can be accessed using array.shape[0] which gives row count
- `size` → total number of items e.g. 2-dim array with shape of (6, 5) will have value of 30
- `dtype` → data type of all data items

6.2 Creation of Numpy Arrays

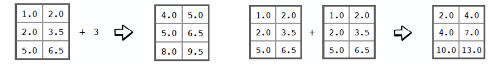
- Creating a 1d array: `array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])`
- Creating a 2d array: `array_2d = np.array([[18, 26, 17], [25, 15.5, 12], [24, 27, 20], [10, 5.5, 17], [27, 26, 15], [22, 21, 21]])`
- Creating array with all 1s: `ones_1d = np.ones(5) # returns array([1., 1., 1., 1., 1.])`
- Creating array with all 0s: `ones_1d = np.zeros((3, 4)) # returns array([[0., 0., 0., 0.], [0., 0., 0., 0.], [0., 0., 0., 0.]])`
- Creating a range of floating point numbers: `range_array = np.arange(2, 5, 0.5) # returns array([2. , 2.5, 3. , 3.5, 4. , 4.5])`
  - Note that while `range()` creates a **range** object, `arange()` creates a **ndarray** object
  - `range()` can only **create sequence of integer** while `arange()` can **create sequence of any real number**

6.3 Indexing and slicing

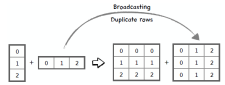
- 1d array follows normal slicing and indexing of arrays
- 2d array could be sliced using [row, col] e.g. `array_2d[3:5, 1:]` will give rows 3,4 and col 1 to the last col, `array_2d[[0, 2, 1], 1]` will give row 0,2,1 and col 1 to the last col

6.4 Vectorized Operation

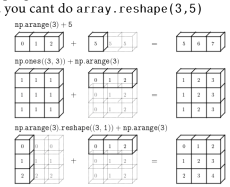
- Similarly to pandas dataframe, we can run element-wise arithmetic operations on them
- Allows for code to be more concise, easier to read and executes faster
- Possible to perform arithmetic operations on 2 numpy arrays with same shape too



- Possible to perform arithmetic operation on 2 arrays with different shape by doing broadcasting
  - Note that broadcasting only works with you are duplicating a **1d array** (either row / col) and does not work with n-dim array (n row / col)



- When operating on arrays with 2 different shapes, it is also possible to use the `reshape()` method to change the shape of the array
  - for e.g. if you had an array of 10 elem, you could call `x_array.reshape(2, 5)` to change the shape of array to a shape of 2 rows, 5 cols
  - Note that reshaping **must maintain the size of the original array**, i.e. if original array 10 elems, you cant do `array.reshape(3,5)`



6.5 Numpy methods

- `np.log(<arr or num>)` → logs each value in array
- `np.exp(<arr or num>)` → gets value of e^i for $i \in$ arr
- `np.square(<arr or num>)` → get value of i^2 for each $i \in$ arr
- `np.power(<power>, <arr or num>)` → gets the value of i^{power} for each $i \in$ arr
- `np.cos()/sin()`
- Supports all of `sum()`, `max()/min()`, `mean()`, `var()/std()`, `var()/std()` calculates the **population var or std** by default
- Can specify the axis to aggregate by passing as args, `axis=0` is col while `axis=1` is row

7 Review of Probability

7.1 General Probability Rules

- Rules of complement: $P(A) = 1 - P(A^C)$, where A^C is the complement of A
- General addition rule: $P(A_1 \text{ or } A_2) = P(A_1) + P(A_2) - P(A_1 \text{ and } A_2)$, if A_1 and A_2 are mutually exclusive then $P(A_1 \text{ or } A_2) = P(A_1) + P(A_2)$
- Conditional Probability: $P(A|B) = \dfrac{P(A \text{ and } B)}{P(B)} \rightarrow P(A \text{ and } B) = P(A|B)P(B)$, if A and B are independent then $P(A \text{ and } B) = P(A)P(B)$

7.2 Discrete Random Variables

- Refers to variables with possible outcomes being finite are countable
- e.g. Preference of coke or pepsi, result of dice roll, number out of x people who prefer Coke over Pepsi (**binomial**), The number of patients arriving in an emergency room within a fixed time interval (**poisson**)

7.2.1 Probability Mass Function (PMF)

Point Probability. Given as $P(X = x_j) = p_j$ for each $j = 1, 2, \cdots, k$ where p must satisfy

$$\begin{cases} 0 \le p_j \le 1 & \text{for each } j = 1, 2, \cdots, k \\ \sum_{j=1}^{k} p_j = 1 \end{cases}$$

7.2.2 Binom example

```
from scipy.stats import binom
x = 8 # test variable, could be an array
n = 10 # size of sample
p = 0.65 # probability of event happening

pmf = binom.pmf(x, n, p)
```

7.3 Cumulative Distribution Function (CDF)

The CDF of a random variable X is defined as $F(x) = P(X \le x), F(x) \le 1$. Can be understood as the probability of an event happening $\le X$ times. In discrete variables, graph would typically look like a staircase

- In questions that asks "what is the probability that at least x ...", we could either do `1-<distribution>.cdf(x-1, p)` OR `<distribution>.cdf(x, 1-p)`

7.4 Continuous Random Variables

- Refers to variables where it could take all values in an interval of numbers

> **Notes:** Let $F(x) = P(X \le x)$ be the CDF of a continuous random variable X, then
> - The derivative $f(x) = \dfrac{dF(x)}{dx}$ of the CDF $F(x)$ is called the **probability density function (PDF)** of X. This definition also implies that $F(x) = \int_{-\infty}^{x} f(t)dt$.
> - The inverse of CDF $F(x)$, denoted by $F^{-1}(q)$, is called the **Percent Point Function (PPF)**, where q is the given cumulative probability. This function is sometimes referred to as the **inverse cumulative distribution function** or the **quantile function**.

> **Notes:** For a continuous random variable X and given values $x_1 \le x_2$, then the probability
> $$P(x_1 \le X \le x_2) = P(X \le x_2) - P(X \le x_1) = F(x_2) - F(x_1).$$

- Note that for Continuous Random Variable, there is no concept of $P(x_1 = X)$ and $P(x_1 = X)$ is always $= 0$

7.4.1 Normal example

```
from scipy.stats import norm
x = -30
mean = 20.6
std = 30.85

pmf = norm.cdf(x, mean, std)
```

7.4.2 Percent Point Function

- Could be understood as asking: What is the actual value that will cause x_1 to occur with a probability $\ge X$

7.5 Expected Value and Variance

	Discrete	Continuous
$\mathbb{E}(X)$	$\sum_{i=1}^{k} x_i p_i$	$\int_{x \in \mathscr{X}} x f(x) dx$
$\text{Var}(X)$	$\sum_{i=1}^{k} (x_i - \mathbb{E}(X))^2 p_i$	$\int_{x \in \mathscr{X}} (x - \mathbb{E}(X))^2 f(x) dx$

Distribution	Parameters	Expected value	Variance	SciPy object	Remarks
Binomial	n as a positive integer $0 < p < 1$ as a probability	np	$np(1-p)$	`binom`	-
Poisson	$\mu > 0$	μ	μ	`poisson`	-
Uniform	a as the lower bound b as the upper bound	$\frac{a+b}{2}$	$\frac{1}{12}(b-a)^2$	`uniform`	$a = 0$ and $b = 1$, by default
Normal	μ as the mean value $\sigma > 0$ as the standard deviation	μ	σ^2	`norm`	$\mu = 0$ and $\sigma = 1$, by default

7.5.1 Properties of Expected Value and Variances

- $\mathbb{E}(c) = c$
- $\mathbb{E}(aX + c) = a\mathbb{E}(X) + c$
- $\mathbb{E}\left(\sum_{i=i}^{n} a_i X_i\right) = \sum_{i=1}^{n} a_i \mathbb{E}(X_i)$
- $\text{Var}(c) = 0$
- $\text{Var}(aX + c) = a^2 \text{Var}(X)$, Note that $Cov(X, X) = Var(X)$
- $\text{Var}(aX + bY + c) = a^2\text{Var}(X) + b^2\text{Var}(Y) + 2ab\text{Cov}(X, Y)$, if all random variables are **pairwise independent**: $\text{Var}\left(\sum_{i=1}^{n} a_i X_i\right) = \sum_{i=1}^{n} a_i^2 \text{Var}(X_i)$

7.5.2 Log returns

> **Example 7:** The **log return**, denoted by r_t, is defined as
> $$r_t = \log\left(\frac{Q_t}{Q_{t-1}}\right),$$
> where Q_t and Q_{t-1} are prices of an asset at time t and $t-1$, respectively, and the $\log(\cdot)$ is the natural logarithm function. Suppose that the daily log returns on a stock are independent and normally distributed with mean 0.001 and standard deviation 0.015.
> - If you buy \$1000 worth of this stock at time $t = 1$, what is the probability that after one trading day, i.e. at time $t = 2$, your investment is worth less than \$990?

$$P(Q_2 \le 990) \Rightarrow P\left(\frac{Q_2}{Q_1} \le 0.99\right) \Rightarrow P(r_2 \le \log(0.99))$$

- One nice property of log returns is that regardless of what t is, the returns will always be $\log\left(\dfrac{Q_t}{Q_1}\right)$

```
mean_z = mean * 5
std_z = std * (5**0.5)

prob = norm.cdf(np.log(0.99), mean_z, std_z)
print(f'The probability is {prob}')
```

```
The probability is 0.3268188763247845
```

$$z = r_2 + r_3 + \ldots + r_6 \Rightarrow \begin{cases} \mathbb{E}(z) = \mathbb{E}(r_2) + \mathbb{E}(r_3) + \ldots + \mathbb{E}(r_6) \\ \text{Var}(z) = \text{Var}(r_2) + \text{Var}(r_3) + \ldots + \text{Var}(r_6) \end{cases}$$

8 Random Sampling

- **Population**: the collection of all individuals or items under consideration in a statistical study → impractical to study whole population due to **time and cost constraints**
- **Sample**: part of the population from which information is obtained

8.1 Population parameters and Sample Statistics

typically use this instead of population statistics since data easier to collect

	Population Parameters	Sample Statistics
mean value	population mean μ	sample average $\bar{X}$
standard deviation	population standard deviation σ	sample standard deviation s
Probability	population probability p	sample proportion $\hat{p}$
Distribution	PDF or PMF	histogram or density plot

	Population	Sample
Mean	$\mu = \begin{cases} \sum_{i=1}^{k} x_i p_i \\ \int_{x \in \mathscr{X}} x f(x) dx \end{cases}$	$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$
Variance	$\sigma^2 = \begin{cases} \sum_{i=1}^{k} (x_i - \mathbb{E}(X))^2 p_i \\ \int_{x \in \mathscr{X}} (x - \mathbb{E}(X))^2 f(x) dx \end{cases}$	$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})^2$

8.2 Central Limit Theorem

- For a relatively large sample size, the random variable $\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$ is approximately normal distributed, regardless of distribution of population. Approximation becomes better with increased sample size.

9 Confidence Interval and Hypothesis Testing

9.1 Review of Sampling Distribution

- $\mathbb{E}(\bar{X}) = \mathbb{E}\left(\frac{1}{n} \sum_{i=1}^{n} X_i\right) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}(X_i) = \frac{1}{n} \sum_{i=1}^{n} \mu = \mu$
- $\text{Var}(\bar{X}) = \text{Var}\left(\frac{1}{n} \sum_{i=1}^{n} X_i\right) = \frac{1}{n^2} \sum_{i=1}^{n} \text{Var}(X_i) = \frac{1}{n^2} n\sigma^2 = \frac{\sigma^2}{n}$

9.2 Confidence Interval

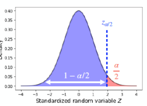
- **General idea**: There are many *point estimates around the population mean* and each of the point estimates can *exist in a range of plausible value* (depending on sample or population std)
- **Confidence Interval** is given by the equation: `estimate ± margin of error`
  - Note that if population std is known, **margin of error will be the same for each point estimate**

9.2.1 Confidence Interval when σ is known

- $\bar{X}$ is approximately normally distributed (by CLT)
- Mean value of $\bar{X}$ is the population mean μ
- Std of $\bar{X}$ is $\sigma/\sqrt{n}$
- z-value: $\dfrac{\bar{X} - \mu}{\sigma/\sqrt{n}} \sim N(0, 1)$
- **Interval**: $\bar{X} \pm \dfrac{\sigma}{\sqrt{n}} \cdot z_{\alpha/2} \rightarrow$ Alt Upper bound: `x_bar-norm.ppf(alpha/2)*se`
- $\dfrac{\sigma}{\sqrt{n}}$ is known, only need to calculate $z_{\alpha/2} \rightarrow F^{-1}(1 - \alpha/2)$ (PPF of $\alpha/2$)
- Code: `z_alpha2 = scipy.stats.norm.ppf(1-alpha/2)`

z-value: $Z = \dfrac{\bar{X} - \mu}{\sigma/\sqrt{n}} \sim N(0,1)$

$P(Z \le z_{\alpha/2}) = F(z_{\alpha/2}) = 1 - \alpha/2$ cdf of $z_{\alpha/2}$



9.2.2 Confidence Interval when σ is unknown

- t-value: $T = \dfrac{\bar{X} - \mu}{s/\sqrt{n}} \sim t$−distribution → tends to Z−distribution as n increases (Degree of Freedom (n-1) increases)
- **Interval**: $\bar{X} \pm \dfrac{s}{\sqrt{n}} \cdot t_{\alpha/2}$
- Code: `t_alpha2 = scipy.stats.t.ppf(1-alpha/2, n-1)`

9.3 Confidence Interval for Proportions

- Population proportion: p, Sample proportion: $\hat{p} = \frac{m}{n}$
- $\mathbb{E}(\hat{p}) = \mathbb{E}\left(\frac{m}{n}\right) = \frac{\mathbb{E}(m)}{n} = \frac{np}{n} = p$
- $\text{Var}(\hat{p}) = \text{Var}\left(\frac{m}{n}\right) = \frac{\text{Var}(m)}{n^2} = \frac{np(1-p)}{n^2} = \frac{p(1-p)}{n} \rightarrow \text{Var}(\hat{p})$ decreases as n increases
- Shape of sampling distribution approaches normal as n increases
- **Interval**: $\hat{p} \pm \sqrt{\dfrac{\hat{p}(1-\hat{p})}{n}} \cdot z_{\alpha/2} \rightarrow$ max margin of error is when $\hat{p}(1 - \hat{p}) = 0.5$

9.4 Hypothesis Testing

- 3 types of tests: 2-tailed test - $H_a : \mu \ne \mu_0$, Left-tailed - $H_a : \mu < \mu_0$, Right-tailed - $H_a : \mu > \mu_0$
- General steps:
  ```
  n = <no. of samples>
  alpha = <significance level>
  mu0 = <null hypothesis>
  t_value = (x_bar - mu0) / (sample_std / n ** 0.5)
  ```
- If right-tailed test: `p_value = 1 - t.cdf(t_value, n-1)`
- If left-tailed test: `p_value = t.cdf(-t_value, n-1)`
- If 2-tailed test: `p_value = 2 * (1 - t.cdf(t_value, n-1))` OR `p_value = 2 * (t.cdf(-t_value, n-1))`
- If p_value > alpha, **reject alternative hypothesis** in favor of null hypothesis
- If p_value < alpha, **reject null hypothesis** in favor of alternative hypothesis